QLToday

Volume 14
Issue 3
March - May
2010

ISSN 1432-5454

The Magazine about QL, QDOS, Sinclair Computers, SMSQ...

Q-emuLator 3.0 for Windows

by Daniele Terdina

Easy PEasy

by Norman Dunbar

Basic Programming Aids

by Dylwin Jones

Lucerne "QL & Mac are 25" show

The lost treasures in-depth

by Urs König

12 Extra pages!

www.QLToday.com

Contents

The deadline for the next issue is the 5th of May 2010

Advertisers

in alphabetical order

As we filled 44 pages this time (instead of the scheduled 32), no space was left for the JMS ad. As I print the address on a separate sheet, I decided to put the JMS ad onto the reverse side of the sheet and kindly ask you to take notice of it.

QL Today

ISSN 1432-5454

German office & Publisher:

Jochen Merz Software
Kaiser-Wilhelm-Str. 302
47169 Duisburg
Germany

Tel. +49 203 502011
Fax +49 203 502012
email: smsq@j-m-s.com
email: QLToday@j-m-s.com

Editor:

Geoff Wicks
Flat 5b
Wordsworth Avenue
Derby DE24 9HQ
United Kingdom

Tel. +44 1332 271366
email: gtwicks@btinternet.com
email: QLToday@j-m-s.com

Co-Editor & UK Office:

Bruce Nicholls
38 Derham Gardens
Upminster
Essex RM14 3HA
United Kingdom

Tel +44 20 71930539
Fax +44 870 0568755
email: qltoday@q-v-d.demon.co.uk
email: QLToday@j-m-s.com

QL *Today* is published four times a year, our volume begins on beginning of June. Please contact the German or English office for current subscription rates or visit our homepage **www.QLTODAY.com**.

We welcome your comments, suggestions and articles. YOU make **QL *Today*** possible. We are constantly changing and adjusting to meet your needs and requirements. Articles for publication should be on a 3.5" disk (DD or HD) or sent via Email. We prefer ASCII, Quill or text87 format. Pictures may be in _SCR format, we can also handle GIF or TIF or JPG. To enhance your article you may wish to include Saved Screen dumps. PLEASE send a hardcopy of all screens to be included. Don't forget to specify where in the text you would like the screen placed.

If you need more information about the UNZIP program which is used by our BOOT program to unpack the files, we suggest that you visit Dilwyn Jones' web site where you find more information about lots of interesting QDOS software and INFOZIP at **http://www.dilwyn.uk6.net/arch/index.html**

Editorial

by Geoff Wicks

Something strange is happening on the mainland of Europe.

The ink was barely dry on my last editorial with the words, "The party is probably over for QL shows" when up popped Gerhard Plavec with a proposal for a show in Austria this year. For several years the Netherlands had appeared to be the last bastion of European mainland shows, but in 2008 there was a show in Italy, last year in Switzerland and this year in Austria. The common factor is that the three lands no longer have a QL user group and instead an individual has taken an initiative.

Paradoxically in the one European land with a huge user group - 175 members at the last official count - QL shows are at best stagnating if not in decline. Is this the way UK QL-ers want it? Whenever Quanta or QL Today has tried to start a discussion on the future of UK shows there has been no response whatsoever. Whenever someone has suggested setting up an internet link at shows - something that has often happened outside the UK - there has been no enthusiasm whatsoever.

Six years ago I met the Quanta committee to discuss the QL2004 international show in Eindhoven. The then chairman rubbished the plans saying Sin-QL-Air was incapable of running an international event and all present nodded their agreement. Quanta turned its back on QL2004 and missed out on the most significant QL event of the last decade. In practice there have been more international QL shows in the Netherlands than in any other land. Perhaps Quanta could still learn from other lands.

At that meeting the members of the committee expressed pride in not subscribing to the QL-users email group and one described its discussions as being petty and puerile. The most bizarre moment came when the chairman rebuked me for using the word "we". I should not presume to have the same status as a member of the Quanta committee.

One of the then officers broke through this way of thinking. He became a regular participant in the QL-user group and, in so doing, gave Quanta a human face. He became the person people instantly turned to when they wanted contact with Quanta. His efforts have brought results. Earlier this year he could boast that all current Quanta committee members are subscribers to the QL-users group. And last year Quanta was not only represented at the continental 25 year celebration, but was also a sponsor of it.

Six years ago many people used the word "aloof" in connection with Quanta. Few would use that word today.

Many people are hoping for some miracle that will save the QL. Occasionally suggestions are made that have an initial attractiveness, but do not stand up to detailed scrutiny. Giving a free copy of QPC2 to all Quanta members would bankrupt the organisation and be an inefficient use of resources. The new notebook PCs are hardly suitable for elderly QL-ers who need larger keyboards and larger screens rather than smaller. A merger between Quanta and QL Today is just not on the cards.

There is no magic formula to save the QL, but what we can see from recent events is that when one individual has a vision, and is prepared to get his hands dirty, he can achieve great things.

There is a future for the QL, but only if we are prepared to work for it.

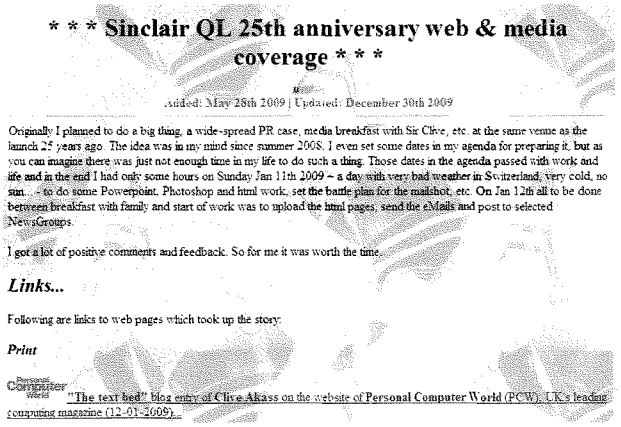
VITAL STATISTICS

Urs König has closed his unique celebration of 25 QL years by releasing some statistics showing the results of his efforts. On one day his website had almost 900 hits and all told his YouTube channel attracted 15,000 viewings.

There were 8,698 unique visitors to the site of which 6,341 viewed the quarter centenary page. Much of the interest was stimulated by third party blogging, one of whom was Linus Torvalds, the initiator of Linux development. About 2,000 visitors came to the site via Torvalds' blog with 891 being on the day following his blog entry. In addition there were 464 downloads of the QPC demo version and 1,301 of the PowerPoint presentation.

Urs has now closed his celebration of the 25th anniversary of the QL, but for full details of the site hits and media coverage of the quarter centenary QL-ers can visit his website:

http://tinyurl.com/ql-is-25



*** * * Sinclair QL 25th anniversary web & media coverage * * ***

Added: May 28th 2009 | Updated: December 30th 2009

Originally I planned to do a big thing, a wide-spread PR case, media breakfast with Sir Clive, etc. at the same venue as the launch 25 years ago. The idea was in my mind since summer 2008. I even set some dates in my agenda for preparing it, but as you can imagine there was just not enough time in my life to do such a thing. Those dates in the agenda passed with work and life and in the end I had only some hours on Sunday Jan 11th 2009 – a day with very bad weather in Switzerland, very cold, no sun... – to do some Powerpoint, Photoshop and html work, set the battle plan for the mailshot, etc. On Jan 12th all to be done between breakfast with family and start of work was to upload the html pages, send the eMails and post to selected NewsGroups.

I got a lot of positive comments and feedback. So for me it was worth the time.

Links...

Following are links to web pages which took up the story:

Print

Personal Computer World "The text bed" blog entry of Clive Akass on the website of Personal Computer World (PCW), UK's leading computing magazine (12-01-2009).

SOFTWARE NEWS

Q-emuLator

Daniele Terdina is planning to release a major upgrade of Q-emuLator containing many new features. Elsewhere in this issue and exclusive to QL Today Daniele writes about the changes he plans.

QL CALENDAR

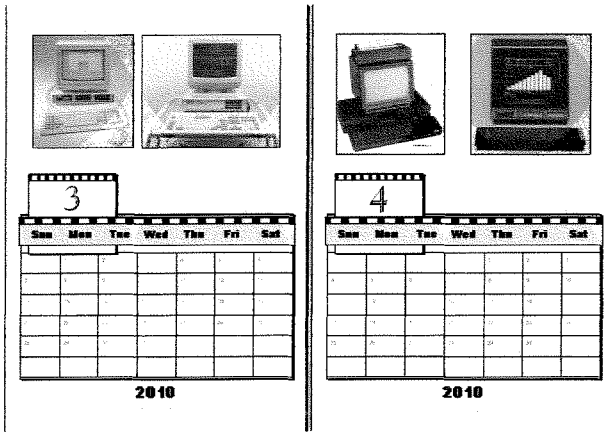
Dilwyn Jones has continued his tradition of designing a QL calendar each year. This year it has images of the QL in its many shapes and forms. The calendar can be downloaded from:

http://www.dilwyn.me.uk/gen/calendar/calendar.html

It is available as a Word.doc (2.6 MB) or as a PDF (1.2MB) file.

Dilwyn also reminds us of the numerous calendar programs and utilities available on his website:

http://www.dilwyn.me.uk/utils/index.html



PROCMAN 2

Dilwyn further announces:

"I have just released version 2 of Procman, a program for extracting procedures and functions from a BASIC program.

This new version is pointer driven and uses Window Manager 2, allowing it to use whatever colour themes you use on your WMAN2 system - WMAN2 is available on SMSQ/E version 3.00 or later, or QDOS with Pointer Environment version 2.00 or later. This version does away with the need to have the Menu Extension present, although it can send the extracted routines to the Scrap system if present so that you can paste the extracted routines into an editor such as QD if you are in the habit of using an editor to write BASIC programs.

Procman 2 can send extracted routines to a file, to screen, to printer, to Scrap, or even direct to BASIC if required.

And thanks to helpful information and routines from Norman Dunbar and Per Witte, Procman 2 can now extract routines from _sav (QSAVEd) files too.

The way in which the program now operates makes it even easier and faster to use!"

Procman 2 is freeware, and can be downloaded from the Programming Utilities page on Dilwyn's website:

http://www.dilwyn.me.uk/program/index.html

SHOW CALENDAR

Later this year Gerhard Plavec is planning an international QL show in Prottes near Vienna. It

will be held from Thursday 3rd to Sunday 6th June with the main events provisionally planned for the Saturday.
Details in German, French and English can be found on Gerhard's website:
http://kuel.org

Das internationale QL-Treffen 2010 in Prottes

La rencontre internationale 2010 à Prottes

The international QL meeting 2010 in Prottes

maps.google.com

Quanta has announced that its AGM and Workshop will be held on Sunday 18th April at a Retirement Centre in Birmingham. The address is "Fircone", 1237 Stratford Road, Hall Green, Birmingham, B28 9AA. Quanta reports the building has a large car park and is within a short walking distance of Hall Green railway station.
Quanta has ended a long tradition of rotating shows between north and south of England on alternate years, but is now opting, as last year, for a central location.

Sin_QL_air is hoping to hold two meetings in Eindhoven - one at the end of April and the other at the beginning of October, but this is dependent on the health of the organiser.

NEW RETRO WEBSITE

As briefly mentioned as a late news item in the last issue Rich Mellor has now launched a new retro auction website.
Rich writes:
"The idea is to bring traders, collectors and users together - we have added a wiki to each category and could do with some more wiki text (or different languages).
If we have missed any categories, please let us know.
The fees are a lot cheaper than on ebay, and we are working towards two types of webstore - a free webstore linked to a category, so that traders can let people know where else that they can try, or a paid for webstore (£5 per month) with 50% off all fees on the site.
Plenty more ideas in the offing, but we are open to suggestions!"

WEB ADDRESS CHANGE

The Just Words! website will shortly have a new address:
http://members.multimania.co.uk/geoffwicks/justwords.htm
This is not a new host, but is the latest development in the hosting transfer of the Tripod sites. Currently automatic redirection from the former Lycos host is taking place, but this will shortly be ended.
Just Words! is still reviewing the short and long term future of the site and is currently analysing traffic patterns on the web pages. There has been a recent increase in visitors to the site, but there is evidence that about one tenth of the visitors are non-QL-ers who have come to the site via search engines.
Just Words! is unhappy with the present site host and options for the future range from the complete closure of the site to its expansion. Just Words! believes the QL community is currently in a state of flux in which major developments could occur in the near future. Should, in the light of these developments, Just Words! finally decide the website could have a long term future then a move to a more secure host is planned.

NEW SURVEY

Quanta is holding a new survey of QL use by both members and non-members. This can be completed online at:
http://quanta.nikad.net//index.php?sid=88977&lang=en
The last survey was in 2004 when non-members were also invited to participate. Over a quarter of Quanta members responded to that survey and 42% of the replies came from non-members. To encourage a good response in 2004 Quanta awarded a number of lottery prizes, one group for members and another for non-members, but this time no lottery is being held.
The new survey hit early teething problems with a question in which respondents were asked to rate various features in the Quanta Magazine on a scale of 1 to 5, but which omitted to define

what these figures represented. Early reactions indicate that respondents had interpreted the scale in different ways. Quanta has ignored suggestons that the survey should be restarted thus invalidating any of its findings about the Quanta Magazine.

One surprising finding of the original survey was a high number of "black box" users who made only a simple use of the QL . Some of these users even worked with unexpanded machines and microdrives. They expressed some anger that they were a neglected group in the QL community, but they turned down Quanta's suggestions for help and came with no alternatives of their own.

Major changes will be taking place in Quanta this year as the organisation says John Gilpin must step down according to clause 5.2 of the constitution having served 6 years continuously on the committee. During his period in office John has become the linchpin of the organisation playing a key role in transforming the Quanta Magazine and launching the electronic version. He has also tripled up as treasurer and membership secretary.

Quanta's six year rule is not absolute and is governed not only by clause 5.2, but also clauses 5.3 and 5.4. Clause 5.2 begins with the words "Save as provided in clause 5.3 below..." which indicates unequivocally that there are exceptions to the rule. How the rule applies to officers is defined in clause 5.4. Astute members of Quanta will also spot a weakness in the wording of clause 5.4 that could cause problems in 2012.

Gee Graphics - Part 48

by H.L. Schaaf

Flattered to see my name in print[1] again.

Can't believe Steve didn't find lots of info on the web, perhaps he was trawling with an open net? I found way too much, most of which I hope to 'get back to'.

The best answer I've found to 'true' randomness was Brownian motion in 'a nice cup of hot tea' as described in Chapter 10 of "The Hitchhiker's Guide to the Galaxy" by Douglas Adams.

At the end Steve invited us to do a better digits of Pi generator. For your enjoyment I present a QL version of the 'spigot'[2] algorithm. The spigot algorithm followed the work on digits of e by Sale[3]. Both of these algorithms will give results in the radix of your choice, so you can have decimal, octal, hexadecimal, etc. answers. Call the procedure 'decimalize' for a quick check of the first few digits.

To 'see' how random the digits of PI are, I'll use QL turtle graphics to do a random walk. Really more of a random wander.

If you generate decimal digits, using a radix of 10,100,1000, or 10000; then you will have an option to call the 'wander' procedure. The most likely probability for distance wandered away from the origin is the square root of the number of steps taken, plus or minus a standard deviation of about 0.44 units. So if you choose 100 digits the wander will be about 10 4.4, the most likely value is shown as a circle. Digits of Pi are in the right window, and an equal number of QL random steps are shown in the left window. Do they look 'Brownian' enough for you?

Enjoy.

More later on 'pseudo'-random number generators that we can do with our QL, and some discussion of how SMSQ/E might do it.

References:

[1] Steve Poole: "Random & PI"
QL Today Volume 14, Issue 2, page 37

[2] Stanley Rabinowitz, Stan Wagon:
"A Spigot Algorithm for the Digits of PI"
American Mathematical Monthly, March 1995
Volume 102, Issue 3, pages 195-203

[3] A.H.J. Sale: "The Calculation of e to many significant digits"
Computer Journal, Volume 11, 1968, pages 229-230

Jeremy Gibbons of the University of Oxford has written of a spigot method for Pi that does not require allocating an array beforehand, which he describes as 'unbounded'.

LISTING SpigotWander_bas

```
100 REMark SpigotWander_bas
110 REMark HL Schaaf Feb 12, 2010
120 REMark for QL Today, GG#48
130 :
140 REMark ref: Rabinowitz and Wagon 1995
150 :
160 WTV : PAPER 0 : INK 7 : CLS
```

```
170 :
180 INPUT "radix ? ";radix
190 pow = LOG10(radix)
200 nines$ = radix -1
210 IF radix›10 : nines$ =CHR$(54 + radix)
220 IF (pow=INT(pow))THEN
230  nines$ = radix - 1
240 END IF
250 :
260 INPUT 'digits wanted ? '; dw
270 REMark allocate array
280 da = (11*dw DIV 3)
290 DIM a(da)
300 FOR j = 1 TO da : a(j) = 2 : END FOR j
310 PRINT 'created array of ';da;" 2's"
320 :
330 pi$ = ''
340 nines = 0
350 predigit = 0
360 start = DATE
370 :
380 FOR j = 1 TO dw/pow + 10
390  PRINT #0;'.›';
400  PRINT #0!!j,pi$
410  IF j=3 THEN
420   IF (radix=2) AND (pi$='11') : pi$ =
      pi$&'.':PRINT '.';
430   IF (radix=3) AND (pi$='10') : pi$ =
      pi$&'.':PRINT '.';
440   IF (radix=4) AND (pi$='3' ) : pi$ =
      pi$&'.':PRINT '.';
450  END IF
460  q=0
470 :
480  FOR i = da TO 1 STEP -1
490   x=radix*a(i) + q*i
500   a(i) = x - INT(x/(2*i -1))*(2*i-1)
510   q = INT(x / (2*i -1))
520  END FOR i
530 :
540  a(1) = q - INT(q/radix)*radix
550  q = INT(q / radix)
560 :
570  IF q == (radix-1) THEN
580   nines = nines + 1
590  ELSE
600   IF q == radix THEN
610    predigit = 1 + predigit
620    IF (predigit ‹ radix): d$= predigit
630    IF (pow‹›INT(pow)) AND (radix›10)
       AND (predigit›9) THEN
640     d$ = CHR$(55+predigit)
650    END IF
660    IF nines : d$ = d$&FILL$('0',nines)
670    pi$ = pi$&d$
680    PRINT d$;
690    predigit = 0 : nines = 0
700   ELSE
710    d$ = predigit
720    IF pow‹›INT(pow) THEN
730     IF (radix›10) AND (predigit ›9) :
        d$ = CHR$(55+predigit)
740    END IF
750    IF j ›2 THEN
760     IF pow = INT(pow)THEN
770      IF LEN(d$)‹pow : d$ =
         FILL$('0',pow-LEN(d$))&d$
780     END IF
790     pi$ = pi$&d$
800     PRINT d$;
810    ELSE
820     IF (j = 2) AND (radix‹›4)THEN
830      pi$ = predigit&'.'
840      PRINT pi$;
850     END IF
860    END IF
870    d$ = ''
880    predigit = q
890    IF nines THEN
900     d$ = d$&FILL$(nines$,nines)
910     pi$ = pi$&d$
920     PRINT d$;
930     nines = 0 : d$ = ''
940    END IF
950   END IF
960  END IF
970  IF LEN(pi$) › 3+dw  : EXIT j
980 END FOR j
990 pi$ = pi$(TO dw+1)
1000 laptime = DATE-start
1010 PRINT \pi$,\laptime!!'seconds'
1020 decimalize
1030 IF pow = INT(pow)
1040  INPUT "want to see Brownian motion
      ? y/n ";ans$
1050  IF ans$=='Y' : wander
1060 END IF
1070 :
1080 STOP
1090 :
1100 DEFine PROCedure wander
1110  rtnd = SQRT(dw)
1120  WMON
1130  SCALE 3*rtnd,-1.5*rtnd,-1.5*rtnd
1140  SCALE #2,3*rtnd,-1.5*rtnd,-1.5*rtnd
1150  PAPER 0 :CLS
1160  PAPER #2,0 :CLS #2
1170  PRINT dw;" digits of Pi"
1180  PRINT #2;"QL random 0-9"
1190  INK 4 : CIRCLE 0,0,rtnd
1200  INK #2,4 : CIRCLE #2,0,0,rtnd
1210 REMark base = 10:angle = 360/10 = 36´
1220  POINT 0,0
1230  POINT #2,0,0
1240  PENDOWN
1250  PENDOWN#2
1260 REMark skip the decimal point
1270  FOR i = 1, 3 TO dw
1280   TURNTO 36*pi$(i)
1290   TURNTO#2, 36*RND(0,9)
1300 :
1310   FOR j = 1 TO RND(2,8)
1320    k = RND : REMark mix it up a bit?
1330   END FOR j
1340 :
1350   MOVE 1
1360   MOVE #2,1
1370  END FOR i
```

```
1380 END DEFine wander
1390 :
1400 DEFine PROCedure decimalize
1410  dradix = radix
1420  IF (pow = INT(pow)) : dradix = 10
1430  sum = 0
1440  dp = '.' INSTR pi$
1450  FOR i = 1 TO LEN(pi$)
1460   pw = dp - i
1470   IF pw = 0 : NEXT i
1480   IF pw >0 : pw = pw -1
1490   IF pi$(i) INSTR('0123456789') THEN
1500    pv = pi$(i)
1510   ELSE
1520    pv = CODE(pi$(i))-55
1530   END IF
1540   pv = pv*dradix^pw
1550   sum = sum + pv
1560   PRINT #0;sum!!
1570   IF sum == PI : EXIT i
1580  END FOR i
1590 END DEFine decimalize
1600 :
1610 REMark End of listing
```

"Why reinvent?", there are lots of web resources with billions of digits of pi for free download. We do it just for the DIY satifaction?

Spigot works in various bases (radix). I have checked with radix 2, 8, 10, 12, 16, 100, 1000, 1000, 10000. The radix 10000 spits (spigots?) out 4 digits at a time.

QL Firmware Bugs Myths - Part 3 Epilogue

by Tony Tebby

Just before the launch of the QL, Sinclair Research celebrated the 1,000,000th sale of a Spectrum. Few people seem to have noticed that Rick Dickinson was preparing a special gold Spectrum for Timex to celebrate the 2,000,000th Spectrum built. Even allowing for returns that cannot be refurbished, that is a big excess of manufacturing over sales.

A while later, when Sinclair Research was trying to find a white knight to rescue it from its cash flow problems, the accounts showed a stock of £32,000,000. If this was all Spectrums, at conventional stock valuation that makes about 1,000,000 Spectrums. So all that excess manufacturing is in stock - that is not very good production management.

The rescue fell through, but soon afterwards the Sinclair sales department celebrated a major success - they had sold the whole stock of Spectrums to the Swedish distributor for £5,000,000 (at a big profit, they claimed). That accounts for only 10% of the book stock. Where was the rest of the stock?

I had arranged for Sinclair to distribute my first independent products and received monthly statements. Sales were not wonderful and I was talking about this to a retailer who told me that sales were quite good, the previous month he had bought a certain number of units and had sold them all. The number of units he had bought was more than the total in my monthly statement. Hmm.

The following week I phoned the accounts department and my contact said that he would look into it. The next time I phoned, he said he was not allowed to speak to me and hung up.

I contacted the company that packaged the products and they told me that there had been an initial order for 2000 units and Sinclair had just ordered another 1000. The accounts showed that just under 1000 had been sold. Why should a cash strapped company buy in another 1000 units when it still had 1000 units in stock? The figures may be a thousand times smaller, but isn't this where we started off with the Spectrum?

Soon afterwards Sinclair folded. I went to Milton Hall and found Jim Westwood, all alone in the vast building, studying piles of listings. He thought that there should be a lot of stock somewhere but could not trace it. I told him about my phantom stock problem and we set about searching for it in the listings (much simpler than looking for Spectrums with their multiple sources). We found 2000 in, nearly 1000 out, and two units in stock.

I heard many tales of the demise of Sinclair Research with the staff rushing to grab anything that was not bolted down (and quite a lot that was) and scurrying off to stuff their cars with Sinclair Research's liberated assets. Was this where the stock had gone? No. Organised skimming, on a massive scale, had been going on for some years.

The stock skimmed off from Sinclair was not, however, sold off on the black market or at car boot sales. According to one retailer, it was sold at full price to Sinclair's customers, through Sinclair's normal retail outlets, complete with Sinclair guarantees and represented an estimated £50,000,000 in lost revenue for Sinclair over the years.

Any departure from business probity in the QL launch story is very insignificant compared to that.

Q-emuLator 3.0 for Windows

by Daniele Terdina

I've been prototyping new Q-emuLator features since well before the last version (2.5) was released one year ago. Many of these features are now almost complete and will be released later in the year. I think there's enough new meat (and work spent grinding it) that a major version change (3.0) is in order.

I'd like to thank all users that have supported the development of Q-emuLator over the years, and often sent suggestions and bug reports.

The focus of Q-emuLator will continue to be maximizing compatibility with the original hardware, while also supporting some newer developments and continuing ease of use.

Due to space constraints, in this article I'll assume that you are already familiar with emulators (a free Q-emuLator demo is available online), and focus on what's new in version 3.

The general application logic and interface is not changing much, but the main window can now be resized to show a bigger QL screen even when not running in full screen mode. (And talking of full screen, a new implementation will allow it to work with Windows 7.)

Everyday's supercomputing

In the first days of Q-emuLator's 68000 processor emulation engine, I used to run it on a Mac, with a 68000 inside, that had a similar clock speed and was only a few times faster than the QL's CPU. To get acceptable emulation speed despite the overhead of the emulation layer, it was critical to use every possible optimization and trick to get the best performance. Only on the fastest (33 MHz) Macs was it possible to reach the same speed as a QL.

But that was almost 20 years ago, and everybody knows how PCs become faster every year. In the year the QL was launched (1984), the world's fastest supercomputer was the Cray X-MP. By comparison to that machine, one of today's desktop processors like the Intel Core i5 is about 50 times faster (costs less than $10,000,000, too!), and even more processing power may lay hidden in your video card.

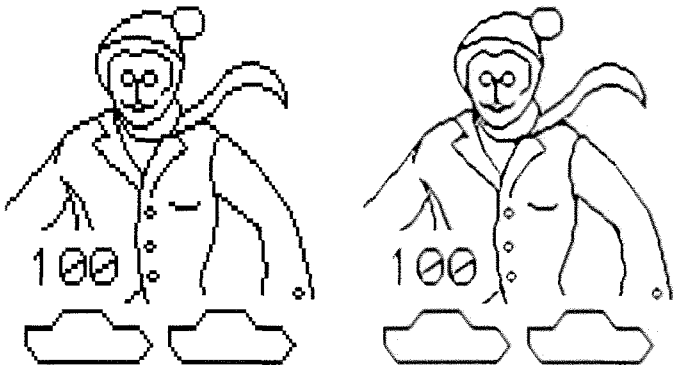
Some of the ideas I've been toying with to further increase emulation speed (dynamic binary translation, quick SuperBASIC, using multiple CPU cores) are interesting but probably unnecessary, and they will not be included in Q-emuLator 3.

Standard daily computer tasks like writing email and surfing the web leave the processor idle for most of its life, so that it gets really bored (OK, maybe not). Operating systems have started to take advantage of the extra available processing power, for example to refine their UI and run background tasks like indexing.

Similarly, running a QL game on an emulator at a speed comparable to that of the original QL (running at maximum speed, it would be up to 500 times too fast to be playable) leaves modern CPUs idle most of the time, so it's becoming possible to take advantage of the unused processing power to improve the emulation experience. Q-emuLator 3 will use the extra cycles for two improvements: accurate emulation speed and increased video quality.

Q-emuLator and other emulators can reduce the emulation speed to benefit software like games. So far, however, the speed has only been an approximation of the real QL's speed, and can be significantly off for many programs. In Q-emuLator 3, the goal is for the 'QL Speed' option to always run at within 5% of the real QL speed.

On today's bigger monitors, the QL's graphics resolutions of 512x256 and 256x256 may look a bit chunky. A user suggestion prompted an investigation on how to increase the display output



quality for more pleasing results when running in full screen mode. The example (see picture on previous page) shows how two pieces of QL graphics (drawings from the games Mortville Manor and TankBusters) look with the current version of Q-emuLator, and in version 3 when the new display filter is enabled.

Managing QL software

Most QL software downloaded from the Internet comes tucked inside .zip files. To run it on an emulator, a user would typically boot in the (virtual) QL with his preferred boot program and use unzip_exe to extract all files and save them to another location like a hard disk. Then he might restart the emulation with different settings to run the new program. (The different settings may be necessary for example because the software is a game that only runs on an unexpanded QL, while unzip needs expanded memory).

To reduce the inconvenience of this process, Q-emuLator will be able to directly mount zip files as read-only drives. Programs that don't need to write to the file system can just run directly from the mounted zip file, while other types of programs can at least be copied to a different drive with a simple WCOPY command.

There will also be a new file format called QLPAK that allows to package QL binaries and supporting data in a single file, together with all the emulator settings necessary for emulation. While creating such files will require a manual process outside the emulator, once they are created using them will be very easy (for example users can double click them in Explorer to execute them). It's a first step toward enabling new possible scenarios, like executing QL programs from the web by clicking on a link, or building an application to manage libraries of QL software (future QLPAKs could contain metadata like copyright notice and a screenshot).

New devices

The file system in Q-emuLator 3 will be able to mount microdrive physical images. You'll be able to mount both .mdv files created for QLay and a new type of .mdv files that can be created on a QL from a real microdrive, using a utility that I'm developing. My hope is that this tool will help to preserve more QL software in its original form even once the original microdrives will no longer be readable. A side benefit is that microdrive images will allow even more software to run on the emulator, as it can now simulate more copy-protection schemes based on the physical image.

Printing from QL software has become an increasing problem. Q-emuLator offers both serial and parallel port devices, but users still need to have a suitable printer and possibly overcome issues with printer configuration and serial cables. In any case, nowadays most printers come with USB connections only. As a solution, Q-emuLator now includes a full printer emulator. Configure your QL software to output to any of the old Epson-compatible printers through the SER port, and the output will instead be converted and redirected to any printer installed on your PC. Just for fun, in addition to the standard mode that provides the best print quality, there is a 'dot matrix' mode that simulates the individual pin strokes as they would be produced by early 9-pin impact printers.

The last of the new devices is for sound output. It's compatible with the 'Sampled Sound System' available on other QDOS systems. You can use the existing driver by Simon Goodwin to send stereo or mono sound samples to the speakers.

Q-emuLator for Mac OS X

Q-emuLator's origins are on the Mac, but the Mac version has not been updated for about 10 years. It's becoming harder to run the old 68000 executable on today's machines, and the emulator itself is in need of a substantial overhaul.

During these years, Apple has radically changed the Mac's OS, UI, processor, file system, services, programming interfaces and language and development tools like compilers and UI designers of the original Mac code, the only part I could reuse was a table of keyboard codes that have uncharacteristically remained unchanged. The new emulator is rather based on a port of the upcoming version for Windows, integrated with an all-new OS X application and user interface.

I won't have time to port all of the Windows features. Priority will be given to emulation of the original QL (for example, precise QL speed emulation), while extra functionality like the TCP/IP driver may not be ported, at least initially.

On the other hand, the fresh start will be a good opportunity to add some polish. For example, the QL display window will be resizable from day one, emulate flashing in 8 colour mode, and make the best of the limited QL's graphics resolution. Q-emuLator for Mac OS X is currently undergoing alpha testing. It should be available for purchase later in the year, and it will only run on Intel-based Macs. OS X 10.5 or later will likely be required.

Easy PEasy - Part 1

by Norman Dunbar

Introduction

At the end of chapter 24, we created a very minimal "Hello World" window using George Gwilt's SETW application. In this chapter, we take a first look at George's other utility to make PE programming easy, EasyPEasy. As mentioned last time, you should have downloaded the `peasp02.zip` file from George's website. If you find a later version (say `peasp03.zip` or higher, get that instead!) The website address is **http://web.ukonline.co.uk/george.gwilt**

Easy PEasy

Unlike many other utilities, Easy PEasy isn't a program you can run, it is a collection of information and small binary files that you can include with your own programs - using the LIB and IN commands in your source code and assembling with GWASL - to make programming the Pointer Environment a little easier. Actually, quite a lot easier as George has done much of the hard work, all we have to do is open a console, make a few checks and write the code to handle our own needs as opposed to the needs of getting the PE up and running.

Much of what follows in this chapter is a blatant theft of George's readme file. For this I make no apology - there is no better way to document something that straight from the horse's mouth!

The Nine Steps To Happiness

With Easy PEasy, there are nine steps to happiness. The following is basically a skeleton for writing a PE program in assembly language:

1. Initialise your program and open a con_ channel.
2. Are the PE & WMAN present? Abort the program if not.
3. Set up the window working definition.
4. Position the window.
5. Draw the window contents.
6. Read the pointer.
7. Did we have an error - exit if so, else it was an event.
8. Process the event.
9. Goto step 6.

Each step in the above, thanks to the coding that George has done, is quite simple.

1. Initialise

The initialisation step consists of setting up your console channel and opening it. The standard QDOSMSQ job header is also required. The code is very simple, and looks like that shown below. It is best to do the set up as soon as possible after the program is executed rather than setting up other stuff first. It saves time and effort - in case something goes wrong and you have to bale out.

```
;------------------------------------------------------------------------------
bra.s     start
          dc.l      0
          dc.w      $4afb

id        equ       0                    Storage (relative A6) for channel id.
wmvec     equ       4                    Storage for WMAN vector. (Relative A6)
slimit    equ       8                    Storage for Window limits call.

jname     dc.w      jname_e-jname-2
          dc.b      "My EPE Program"
jname_e   ds.b      0
```

```
        ds.w      0

; Console definition.
con     dc.w      4
        dc.b      'con_'

; The job starts here.
start   lea       (a6,a4.l),a6          Get the dataspace address in A6.L.

        lea       con,a0                Con_ channel definition.
        moveq     #-1,d1                Required for this job.
        moveq     #0,d3
        moveq     #io_open,d0
        trap      #2                    Open the channel.
        tst.l     do                    Did it work?
        bne       sui                   Exit via sui routine in Easy Peasy.
        move.l    a0,id(a6)             Save the channel id.
;-------------------------------------------------------------------------------
```

2. Check The PE & WMAN

The console is open now, or we have baled out of the program. Obviously we don't get much feedback from the program if anything went wrong, a proper user friendly application would, of course, display a suitable error message. The next easy step is to check for the presence or otherwise of PTR_GEN and WMAN.

The following code requires a channel id, for a CON_ channel, to be in A0.

```
;-------------------------------------------------------------------------------
; Check for PE being present.
;-------------------------------------------------------------------------------
        moveq     #iop_pinf,d0
        moveq     #-1,d3
        trap      #3
        tst.l     d0                    Ptr_gen present?
        bne       sui                   No, bale out.
        move.l    a1,wmvec(a6)          Yes, save WMAN vector
        beq       sui                   Oops! Bale out, no WMAN.
        movea.l   a1,a2                 Keep WM vector in A2
        lea       slimit(a6),a1         Storage, 4 words long.
        moveq     #iop_flim,d0          Need maximum size of window.
        trap      #3
        subi.l    #$C0008,(a1)          Less 12 (width) and 8 (height) for safety.
        lea       wd0,a3                Address of window definition from SETW.
        move.l    #ww0_0,d1             Size of working definition from SETW.
        bsr       getsp                 Allocate space & save address in A0.L.
        movea.l   a0,a4                 Save in A4.L too.
;-------------------------------------------------------------------------------
```

The code above checks for the PE being present and if not found, bales out via the code at sui. If the PE is found, the WMAN vector is saved in data space for later use – however, if WMAN is not loaded (but PTR_GEN is) the job will exit via the familiar sui routine. Easy PEasy requires both the PTR_GEN and WMAN files to be loaded in order to create and run
PE programs.

Next up, we find out the maximum size that the con_ channel can grow to. We assume that that code always works – but it may be good practice to check, just in case. The 4 words returned indicate the size and position of the con_ channel, and these 4 words are placed into the job's data space and a small margin is subtracted from the width and height.

3. Set The Window Definition

The window definition is expected to hold a value in wd0 for the size of working definition and status area space. The code above reads the amount of memory required for the window definition (created by SETW and defined in ww0_0) and allocates space in the common heap for our program to use. If this fails, the call to getsp will never return – it exits through the sui code on error.

```
;-------------------------------------------------------------------------------
; Reserve memory for the window working definition.
;-------------------------------------------------------------------------------
        lea       wd0,a3                Address of window definition from SETW.
        move.l    #ww0_0,d1             Size of working definition from SETW.
        bsr       getsp                 Allocate space & save address in A0.L.
        movea.l   a0,a4                 Save in A4.L too.
;-------------------------------------------------------------------------------
```

If the memory allocation worked, the address is returned in A0 and we save it in A4 for later use. This is a handy feature of Easy PEasy and the way it was written by George.

Before we can call the WMAN routine to set up our window – wm_setup – we need to make sure that the status area for loose items is all initialised properly. The following code assumes that all loose items will be available when the program starts. Zero is the value we need for available.

The labels wst0 and wst0_e are defined by the SETW program. (As you can see SETW does most of the hard work of calculating various sizes and labels for us!)

```
;-------------------------------------------------------------------------------
; Preset all Loose Items to available.
;-------------------------------------------------------------------------------
        movea.l   id(a6),a0             Restore channel ID.
        moveq     #wst0_e-wst0-1,d1     Size of status area + loose items - 1.
        lea       wst0,a1               Wst0 is the address of the status area.

loop    clr.b     (a1)+                 Zero = Loose Item is available status.
        dbra      d1,loop               Clear entire area.
        lea       wst0,a1               Reset pointer to status area.
        moveq     #0,d1                 Default window size.
        lea       wd0,a3                Wd0 is the address of the window definition.
        jsr       wm_setup(a2)          Create the working definition.
;-------------------------------------------------------------------------------
```

4. Position The Window

This is probably one of the easiest parts of the code! We assume that the pointer is in the position on screen where we wish the window to appear. The position of the window may move to make sure that it remains on the screen, however, in normal circumstances, the pointer in our window will be positions in exactly the same place where the on screen pointer is now.

This can be useful if a window has no "move" abilities – you can simply put the pointer where you wish the window to appear, and execute the program. The window will be drawn exactly (adjusted to fit on screen) where you have put the pointer.

```
;-------------------------------------------------------------------------------
; Position, but do not draw, the window.
;-------------------------------------------------------------------------------
        moveaq    #-1,d1                Position window to current pointer position.
        jsr       wm_prpos(a2)          Position a primary window.
;-------------------------------------------------------------------------------
```

5. Draw The Contents

The window has been positioned, however, it has not been drawn on screen, so we need to draw it now. This is even simpler than the positioning of the windows.

```
;-------------------------------------------------------------------------------
; Draw the window.
;-------------------------------------------------------------------------------
        jsr       wm_draw(a2)          Draw the window and its contents.
;-------------------------------------------------------------------------------
```

That was difficult! ;-)

6. The Pointer Loop

At this point, we have the windows on screen and the user is waiting to use the application. We have to enter a loop to read the pointer and act accordingly.

```
;-------------------------------------------------------------------------------
; Main pointer reading loop.
;-------------------------------------------------------------------------------
read_ptr jsr      wm_rptr(a2)          Read the pointer. Will not return here
;                                      Return when either D0 or D4 and non-zero.
;-------------------------------------------------------------------------------
```

For most loose items, application windows and so on, an action routine will have been defined and coded. These action routines will be discussed later. The pointer reading routine - wm_rptr - will not return until either D0 or D4 are non-zero as a result of an action routine.

7. Error Or Event?

If D0 is non-zero, and error has occurred and we should (somehow) handle it and probably bale out of the program. Alternatively, we can simply ignore errors and try again. The program developer decides.

If D4 is non-zero, an event has occurred and we need to handle that in our code before, possibly, returning to the pointer reading loop again.

An event is defined as a key press such as ENTER while the pointer is not positioned on a loose item or menu item, ESC, F1 (Help) or any of the CTRL+Fn key combinations - SLEEP,WAKE, MOVE or SIZE - but only provided that the key press doesn't select a menu item.

An event can be generated by any of the action routines as well. Within the action routines the programmer has the choice of either handling the action code there and then, or, setting an event in D4 and returning. This will cause the call to wm_rptr to exit and return back to the application where the event can be handled.

Some programmers like to control where and when the action handling code is performed and like to keep it all in the main code, others like to carry out the actions within the action handlers. It's entirely up to the developer - the end user will see no difference whichever method is chosen.

Obviously, how a program handles errors and events is up to the programmer and a generic method can't be given here. However, as an example, the following may suffice.

```
;-------------------------------------------------------------------------------
; Ignore errors.
;-------------------------------------------------------------------------------
        bne.s     read_ptr             Error in D0? If so, ignore it.
;                                      The above assumes there is an option in
;                                      the program to allow the user to EXIT.
;-------------------------------------------------------------------------------
```

8. Process Events.

At this stage in our program, we have returned from reading the pointer (wm_rptr) and no errors have been reported (in D0), so we must have detected an event in D4. We have three choices here - if our action routines should have handled things, then perhaps we should ignore the event and read the pointer again - alternatively, this could be an error and
we should abort the program. The other alternative is that our action routines have set the event in D4, so our code should now process the appropriate event.

As above when trapping errors, there's no "one size fits all" answer and every program should handle events accordingly. The following is an example whereby the events are simply ignored and we return to reading the pointer.

```
;-------------------------------------------------------------------------
; Ignore events.
;-------------------------------------------------------------------------
        bra.s     read_ptr            Event number in D4? If so, ignore it.
;-------------------------------------------------------------------------
```

Obviously, if your code is processing events "outside the action routines" then your own code, to process the appropriate event, would go here, rather than simply ignoring the events.

The event numbers are discussed below in "Loose Item Action Routines".

9. Repeat

Repeat has already been handled above. All we do - in this simple example - is loop back to read the pointer when we hit an error or when any event occurs.

Loose Item Action Routines

There are two kinds of action routines you need to be aware of. Those for loose items and those for application menu items. As we have not yet discussed much for Application Windows or their menu items, they will be discussed later.

An action routine for a loose item is called from within the wm_rptr call, and if the action routine exits with D0 and D4 both set to zero, the wm_rptr call will resume again - in other words, control will not return to your own code just yet.

On entry to a loose item action routine various registers are set with specific parameters:

Register	Description
D1.L	High word = pointer X position, Low Word = pointer Y position.
D2.W	Selection keystroke letter, in its upper cased format, or 1 = Hit/SPACE or 2 = DO/ENTER. D2.W may be an event code if an event triggered this action.
D4.B	An event number - see below - if an event triggered this action routine.
A0.L	Channel id.
A1.L	Pointer to the status area.
A2.L	WMAN vector.
A3.L	Pointer to loose menu item.
A4.L	Pointer to window working definition.

If the loose item was triggered as the result of a selection keystroke, D2.W will hold the uppercased letter code.

If the loose item was triggered as a result of an event, D4.B holds the event code and D2.W holds the event number in the table below.

Event Name - Keystroke	Event number in D4.B	Event code in D2.W
Hit - Space	0	1
Do - Enter	16	2
Esc - ESC	17	3
Help - F1	18	4
Move - CTRL+F3	19	5
Size - CTRL+F4	20	6
Sleep - CTRL+F1	21	7
Wake - CTRL+F2	22	8
Move - CTRL+F3	23	9

In addition to the above, the status of the loose item which triggered the action routine will be set to selected. It is not reset to available on exit, this is your responsibility.

Action routines must exit with D5, D6 and D7, A0 and A4 preserved to the same value that they had on entry to the routine. In addition, the code must set the SR according to the value in D0. A5 and A6 can be used and left at any value by the action routines, while D1 - D3 and A1 - A3 appear to be undefined as to their exit status.

If an error is detected, the routine should exit with an error code in D0 and the SR set accordingly. If the action routine simply wishes to cause wm_rptr to return to the user's code where an event will be processed - rather than processing it in the action routine itself, D4 should be set to the correct event code that the user code should process.

Obviously, if setting D4 with an event number then D4 should be set before D0 otherwise the SR will take on the settings for D4 instead of D0.

If no error was detected by the action routine, and no event is to be returned, both D0 and D4 must be set to zero on exit.

The action routine needs to perform the application specific code to process the loose item that was triggered, however, it must also reset the status of the loose item that triggered the action routine. This can be done as follows.

```
;-------------------------------------------------------------------------
; Action routine code goes here ...
;-------------------------------------------------------------------------
        move.l    d5-d7/a0-a4,-(a7)   Preserve those registers that we need.
        ...                           Do your stuff!
        move.l    (a7)+,d5-d7/a0-a4   Restore registers prior to exit.

;-------------------------------------------------------------------------
; Reset loose item status as part of action routine.
;-------------------------------------------------------------------------
        move.w    ww1_item(a3),d1     Get the loose item number.
        move.b    #wsi_mkav,ws_litem(a1,d1.w)  Request the item be redrawn as
                                      available.
        moveq     #-1,d3              Request a selective redraw of loose items.
        jsr       wm_ldraw(a2)        Redraw selected lose items.
        moveq     #0,d4               No event signalled by action routine.
        moveq     #0,d0               No errors either. (Sets SR correctly too.)
        rts                           Back to wm_rptr, not to user code.
;-------------------------------------------------------------------------
```

The code above could be used as a template for loose item action routines. It begins by preserving the registers that we must preserve plus, it stacks A1 and A2 as well – for added safety, as they will be required in the code to reset the loose item status.

Should you reset the status on entry to the routine or exit? It's up to your code obviously. However, I prefer to do it at the end of the action routine. If the action routine is short and quick, it probably makes no difference. If the routine takes some time – lets say, it's formatting a floppy disc – then it's best to leave it at selected until the format finishes and then reset it. However, it's your choice.

That's all for this issue. Next time, I'll take a deeper look at Easy PEasy and the routines that George has written for us. If we have time and space, we might take a look at an example of its use. See you then.

BASIC Programming Aids - Part 1

by Dilwyn Jones

One of the endearing and enduring features of the QL is the ease with which you can write programs in BASIC. Whether you use SuperBASIC on a QDOS system, or the enhanced SBASIC on an SMSQ/E system such as a Q40, Q60 or QPC for example, it's so easy to knock a fairly basic program together in minutes to do the odd task here and there.

SuperBASIC had a rather fiddly little command called EDIT which let you edit lines of BASIC in a limited manner. Then Toolkit 2 gave us an improved editor called ED, which let us scroll through the BASIC program editing whichever lines we wanted, while seeing adjacent lines at the same time. This same ED command is built into SBASIC and works quite well.

So what I hope to do in this article is take a look at some of the programming and editing utility programs and extensions out there which help to make life a little bit easier for the BASIC programmer by helping us with listing variables, procedures and so on, plus enhancing the editing a little bit in various ways and tidying up listings.

In this article I'll use the wildcard term 'S*BASIC' to refer to either SuperBASIC or SBASIC.

QREF

This is probably one of the oldest and best programming aids around. It is a small toolkit with several extensions for listing names to given channels (which thanks to QL device independent can be a screen window, a printer channel or even a file channel). It was originally sold by Liberation Software (the original authors of the Qliberator compiler) and more recently has been available from other QL software suppliers. It's a great help for listing line numbers in the program in memory which use a given name, for example if you wish to change a name, you can use this to list all line numbers which already use the name, to know where to look for the name to change.

It's really easy to use as long as you don't mind typing in commands. It consists of 5 procedures and a function. The procedures list various type of names used in your programs, be they variable names (QREF_V), procedures and functions (QREF_P), machine code extensions (QREF_M) and so on. If you are content with output to one of the standard windows, just issue a command like QREF_V #1 to send it to window channel #1, or OPEN #3,SER1 : QREF_P #3 : CLOSE #3 to send a list of procedures and functions to a printer connected to SER1, for example. The printout includes a list of line numbers where those names are used. Here's a sample printout (Figure 1), taken from a QREF_P printout of procedure and function names.

```
An_Error     BASIC    Procedure  2550
        2760  4630  6710
Centre_X     Float    Function   20160
       18060 20180
Centre_Y%    Integer  Function   20200
        1460  1630  1880
YN$          String   Function   30000
        900   30100
```

Figure 1 - Sample printout using QREF_P command

Here, we can see from the first line, for example, that the program includes a procedure called An_Error, which is defined at line 2550 and the name is also used in lines 2760, 4630 and 6710. There is a floating point function called Centre_X defined at line 20160 and also used in lines 18060

and 20180. The other two examples shown an integer function and string function as well.

QREF includes a function, which can return a line number where a given name is used. This is very useful for use with the ED command for editing. You can use the value returned by QFIND (a line number) to make ED start editing the program at that line number.

ED QFIND('YN$') would cause the ED command to start editing the BASIC program at the line number at which the YN$ function is defined (line 30,000 in the example of Figure 1. Which is useful in itself, but that's not all that QFIND can do. If you don't give it a parameter, just ED QFIND, it will try to find further line numbers where the name you last specified is used and keep doing that if you want until the last instance of that name in the program.

QREF should work even on an unexpanded QL.

Of all the utilities listed here, this has been probably the most used programming aid in my library of programs over the last 25 years.

BASIC Reporter

This is one of my own offerings, a program which I wrote many years ago to help with BASIC programming. It uses facilities provided by the Turbo Toolkit to extract information from the SuperBASIC data structures.

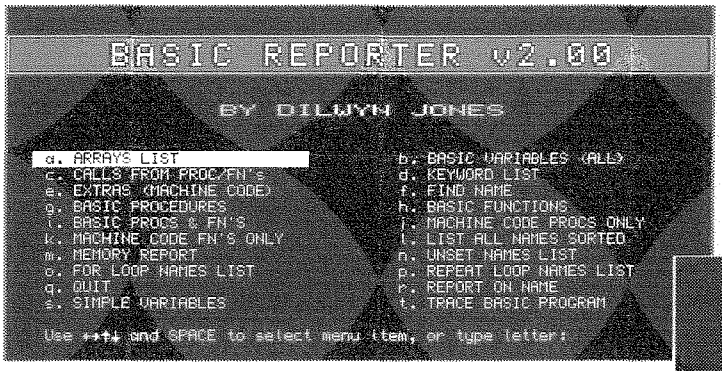


Figure 2 - Basic Reporter main screen

A look at the screen dump of its main screen shows it has about 20 commands you can call up. Some of them are pretty similar to the facilities provided by QREF - listing variables, procedures, functions, extensions and so on. It has a few extra facilities, such as option "c.Calls From Proc/Fn's" which basically does a profile of the calls within a program - a list of which procedures and functions call which other procedures and functions and optionally machine code extensions (see Figure 3 for a sample screen).

As it is a program I wrote myself I won't go an about it at length, the screen dumps should give a reasonable idea of what it can do.

Basic Reporter works on expanded memory systems. Get hold of version 2 (on the same page on my website) if you want to use it on a modern system such as QPC2.

BASIC Reporter is freeware and you can get it from PD libraries or download it from my website at: **http://www.dilwyn.me.uk/program/index.html**

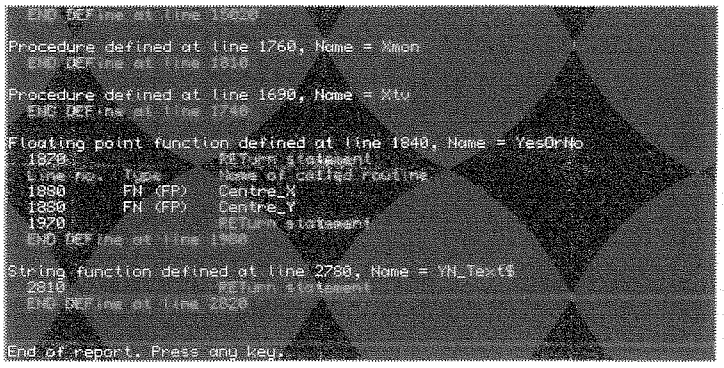


Figure 3 - Calls from Proc/Fns printout

MasterBasic

This is a formerly commercial program by Ergon Development in Italy. Davide Santachiara has now released this program as freeware. It's a Turbo compiled program, with a rather unusual way of working.

For example, on my QPC2 system at least, I run the BOOT program and the opening screen appears, then...nothing. The program vanishes. Or rather, it becomes invisible. A quick check of the JOBS list confirms it's running. Ah well - if all else fails, read the manual.

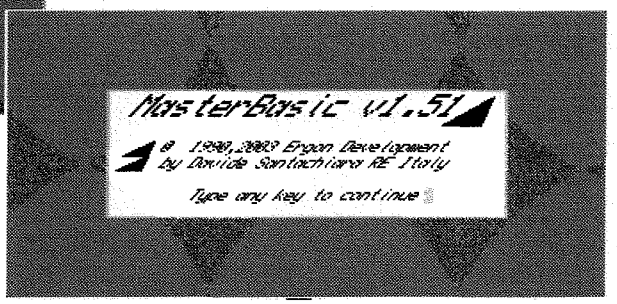


Figure 4 - MasterBasic opening screen

It turns out that a "hidden" set of function key presses is required to bring up the menus. Pressing F1 while you are in S*BASIC brings up the main menu of MasterBASIC - see Figure 5.

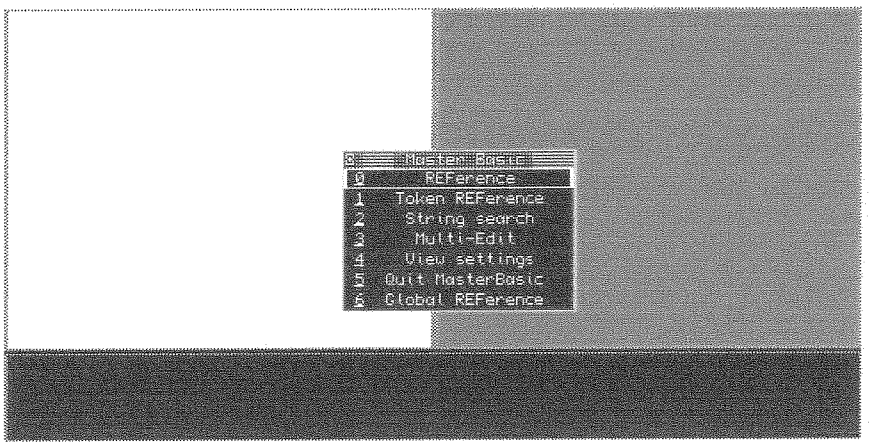


Figure 5 - MasterBasic main menu

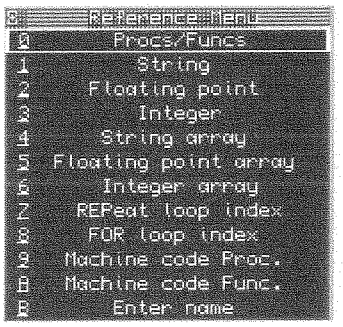


Fig. 6 - REFerence sub-menu

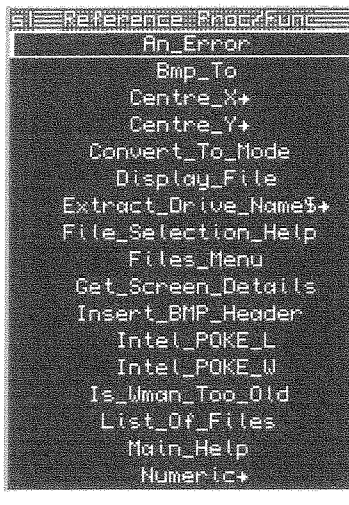


Figure 7 - Sample REFerence list from MasterBasic

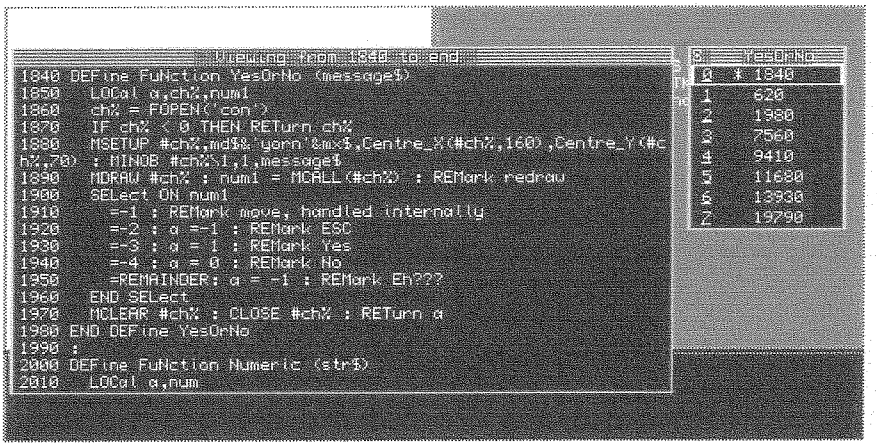


Figure 8 - The Line Numbers list top right, then pressing SHIFT-TAB brings up the listing window

numbers in the current BASIC program which use (or REFerence) that name. Now for the clever bit. Press SPACE or ENTER on one of the line numbers and it will cause the BASIC ED command to start editing at that line number. Or, press TAB to list that one line, or SHIFT TAB to list (view only) from that line, a page at a time. See Figure 8. Several other options are available - the main problem is trying to remember which keypress does what! AT this point I ran into a minor hiccup - I was trying to keep a copy of Xchange Quill open with the instructions and every time I pressed ENTER on a line number to edit the program, it insisted on jumping into Quill instead and inserted the word ED into the document! I'm not sure why this happened but as long as I stuck to using only BASIC and MasterBasic and a printed manual to refer to, it seemed to work OK.

From there it quickly becomes clearer - to select an item move the highlight down to the option required and press ENTER or just press the underlined key. For example, to select REFerence, just press 0. This brings up a sub-menu offering the various types of names which can be referenced (Figure 6).

From this you can select whether to look at procedures, string variables, arrays, machine code extensions etc used in your program. Here's the list I got when I tried it on the sources for my BMP program (Fig. 7)

Scroll through this list and press Enter on the name you want to reference. You'll notice that functions have a mark to the right of the name. This brings up a 'Line menu' which is a list of line

This is certainly a good feature of MasterBasic and although it takes a bit of getting used to, the effort is worth it. It can even work in a Minerva MultiBasic job, although the MasterBasic task has to be owned by the MultiBasic it is editing, in other words, MultiBasic has to be executed from within that MultiBasic. To get it to work properly, read the specific notes in the MasterBasic manual. There are also some notes you need to read regarding the use of MasterBasic within the pointer environment.

A slightly more complex option is the Token REFerence menu. Using this, you can bring up lists of the various tokens (i.e. built in words and symbols understood by S*BASIC, which are

neither procedures nor functions. This does need a bit of knowledge of the S*Basic structures, and the manual refers you to the Jan Jones Super-Basic Guide book if you need to know more.

These Token REFerence searches let you search for other keywords, such as PRINT, WHEN, REPeat and END. You can also search for Operation tokens, such as TO, =, OR and so on. These are all keywords or tokens which aren't defined as explicit procedures or functions. You can also do simple string searches, e.g. you remember that the line you want to edit had a REMark statement with a given piece of text in it, or a string in a PRINT command.

There's also an "accessories" menu accessed by pressing F2 which brings up a menu where you access some simple programming aids like a calculator and a little notepad, plus lists of KEYROW codes and CHR$ codes - press a key and it shows the appropriate KEYROW number or ASCII code.

MasterBasic can be a great programming aid and can achieve results that no other program that I know of can do. It does need a good bit of practice and familiarisation to get used to it before trying to use it in earnest, and you certainly need a copy of the manual to hand at first, but once you get used to it you should find it a great help for editing long, complex BASIC programs.

MasterBasic is available from my Programming Utilities page referred to above

http://www.dilwyn.me.uk/program/index.html

and you can also download it from Davide Santachiara's website's QL software download page at

http://www.sinclairql.it/qlpage.htm

CD and EDIT

This is a pair of extensions to BASIC written by Malcolm Lear. It actually comes as two separate extensions files. The first includes CD and CDp for changing data and program defaults. The other, the one I am interested in for the purposes of this article, is a redefined EDIT command.

The new EDIT command can behave like the old one, but also allows you to specify a string in place of a line number. The string is the name of a procedure or function. EDIT then starts editing at the line at which that procedure or function is defined. Nice and simple - the edit command retains its previous facilities but now also gains a useful new facility.

To install the extension, just LRESPR the file called EDIT_BIN and that's it! Those into assembler can even study the source file, EDIT_ASM.

A nice, neat and simple little aid for the S*BASIC programmer.

It can be downloaded from the Toolkits page on my website at

http://www.dilwyn.me.uk/tk/index.html

Coping with Change

by George Gwilt

In the history of the QL there have been many changes in the operating system, but I would regard only three of these as being major. They are the Pointer Environment (PE), GD2 colours and increased screen sizes. It is interesting that one principle adhered to when these changes gave rise to a new operating system was that existing programs should still work. This principle does not seem to have been applied in Windows, where each new system apparently requires the user to buy yet another version of their favourite software.

Some work has to be done, of course, to ensure compatibility between systems. As an example let's see how PE redefined SD.EXTOP. This trap #3 routine allows the programmer to access the screen driver by means of a piece of code. At entry to this code A0 points to "the channel block". In the absence of the PE this points to the header, which is $18 bytes before the relevant information about the window which starts with the x-position of the upper left corner of the window. The PE has a different version of this channel block. In fact there are $30 more bytes in it. Thus the PE version of this trap sets a value in A0 which is $30 bytes beyond the actual start of the channel so that once again it will point to $18 bytes before the relevant information. This means that SD.EXTOP will work correctly for all programs whether written before or after PE.

After the PE came GD2 colours. How did PE cope? One severe problem was that when each PE program, of standard format, starts it uses a window definition to set up a working definition. This implies that even when operating under new environments, such as GD2, the PE software, which itself can, and indeed must, change, has still to use the original window definition supplied. Either that, or every old PE program has to be al-

tered for the new environment. Clearly that is not acceptable. As it happens, the designers of PE, with immense foresight, before GD2 appeared, set aside, not a byte, but a word for the definition of colour, allowing five bits for each of red, green and blue. This meant that the window definition could easily be used with the new GD2 system.

There is one other aspect of the window definition which is less easy to understand. The definition consists of several parts connected by pointers. These pointers are of word length and contain the distance from the position of the pointer to the address of the part. At some stage it was found, or the possibility was surmised, that the part could be further away from the pointer than 32K bytes. What to do? Presumably the use of two-byte word pointers rather than four-byte long word pointers was chosen because this kept down the size of the window definition. I suppose the decision was made to continue to use word pointers by applying ingenuity. The distance between pointer and target is always an even number of bytes. This means that a true relative pointer always has a zero least significant bit. So, this bit was set to 1 to signal that the target was too far away. In that case the pointer (without the added bit of course) would point to a long word which contained the distance from that long word's position to the target. Problem solved or nearly. Two things result from this. First, the size of the window definition increases by four bytes for each case of indirection. Second, there is a real problem if no long word can be found close enough to the word pointer. Assuming that there are enough long words in the vicinity, the number of bytes of the resulting window definition devoted to the pointers will be

```
2*N + 4*P
```

where there are N pointers and P cases of indirection. If P equals N/2 then there are as many bytes needed for pointers as there would have been if the decision had been taken to have long word pointers in the first place. Bigger values for P mean that using word pointers actually make the window definition longer than it would have been for long word pointers.

Actually in TurboPTR, where each part of a window definition is placed in separate areas allocated from the heap, I had to ensure that each section had a set of long words equal in number to the set of pointers in the section. Thus the length of the window definition needed for pointers in TurboPTR is 6*N.

Oddly enough I recently had to solve a problem in TurboPTR concerning pointers similar to that in PE's window definition. The window definition in TurboPTR contains, not only the sizes and colours of all main windows with their information and application windows, but also the set of non standard sprites, blobs and patterns used in the program. When the window definition is being set up by TurboPTR a file of these sprites, blobs and patterns is loaded into RAM. The make up of the file is this.

The sets of sprites, blobs and patterns consist of a word giving the number of items. This is followed by a set of words pointing to each of the items. Each pointer is the distance from the fourth byte of the file to the item. The first word of the file is a pointer to the set of blobs; the next word points to the set of patterns; then comes the set of sprites. What I found recently was that it is quite easy to have so many sprites that the file is over 32K bytes long. If I had used long word pointers this problem would not have appeared. Having used word pointers I had to live with that, for the same reason that PE could not have changed to long word pointers.

What I did was to notice that my pointers were always even and, also, that they were always positive. When the distance exceeded 32766 I altered the amount by subtracting it from 32766 and dividing it by two. I now had a negative number, fitting into a word and allowing distances up to 98302. To find the true distance implied by one of these negative pointers I simply doubled it and subtracted it from 32766.

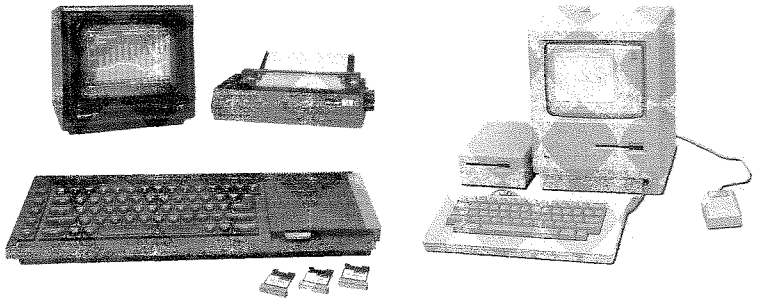
Conclusion

As a final example showing that it is considered important to allow existing programs to continue working under new versions of the operating system I will tell you what happened to a program of mine when I ran it under the then latest version of SMSQ/E. To my surprise the program did odd things with the screen although otherwise it worked well enough. On investigation I discovered that the problem was that the vector UT.MTEXT, which was supposed on exit to leave D3.W equal to -1, didn't. Instead the value of D3 was preserved. I wanted D3 to be -1 for the timeout for subsequent trap #3 calls but, of course, this did not happen.

When I wrote to Tony Tebby about this he immediately made the correction and my program started working normally again. However in discussing this change, he referred to me as Quilt. On being informed that the name was really Gwilt, Tony simply said that he must have been a bit sleepy at the time.

Lucerne "QL & Mac are 25" show
The lost treasures in-depth

by Urs König

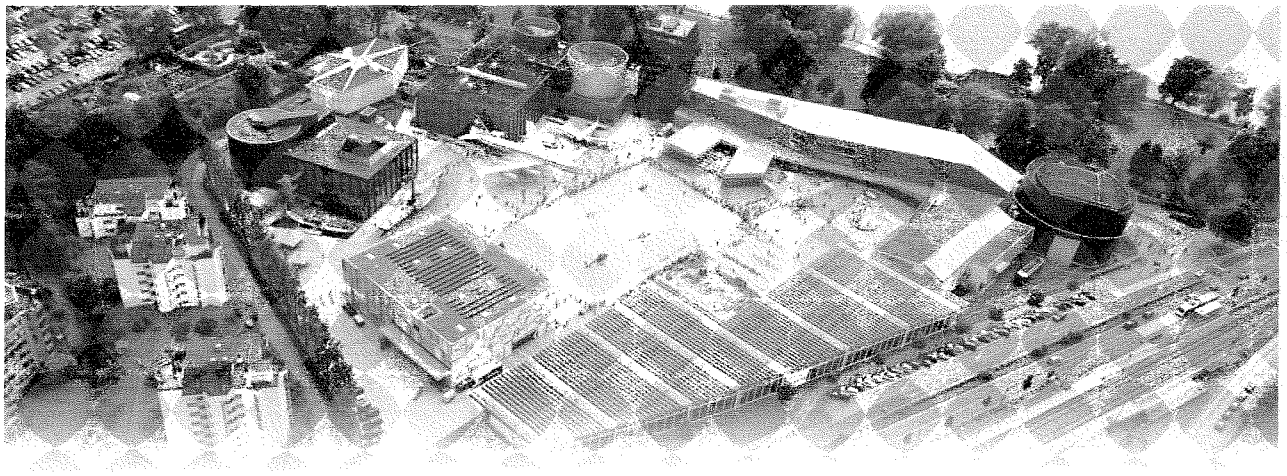


"QL and Mac are 25"
25th anniversary 2009

The show was just one – but the most time consuming – of my activities in the "QL-is-25" bash. I shall summarise the QL 25th anniversary year in detail in a series of three articles in the QUANTA magazine. Tony Firshman and Dilwyn Jones have already reported on the show in general in both QL Today (V14I2) and QUANTA magazines (V26I6). In this article I shall present in-depth some specific QL issues which were part of the show.

Before I start

The show was something I always wanted to do. Due to my change in work/life balance in 2009 I was finally able to do it. Overall I must say that I'm very happy with the happening. The only sour thing is that we did not have many continental visitors. Especially I missed the Italian, French and Croatian QLers to name just a few countries from where travelling to Switzerland would have been less demanding than travelling from the UK. Hey, stop! No more complaints. OK.



In a nutshell

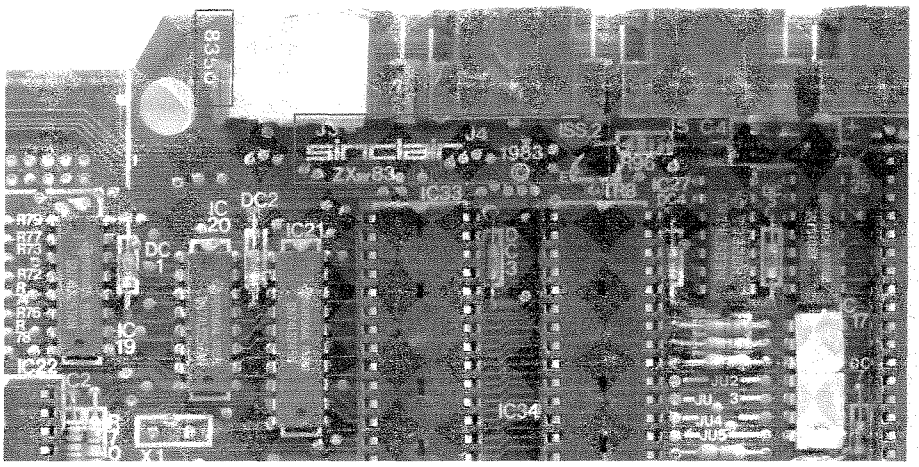
The "QL & Mac are 25" international event took place on Oct 31st/Nov 1st 2009 in the Verkehrshaus, Lucerne, Switzerland (Swiss transport museum). Some 50 people attended the show. Visitors came from the UK, Austria, Germany, Holland and Switzerland. 20 people joined the 25th anniversary dinner which took place in the "Bistro du Théatre" on Saturday Eve.

The lost treasures

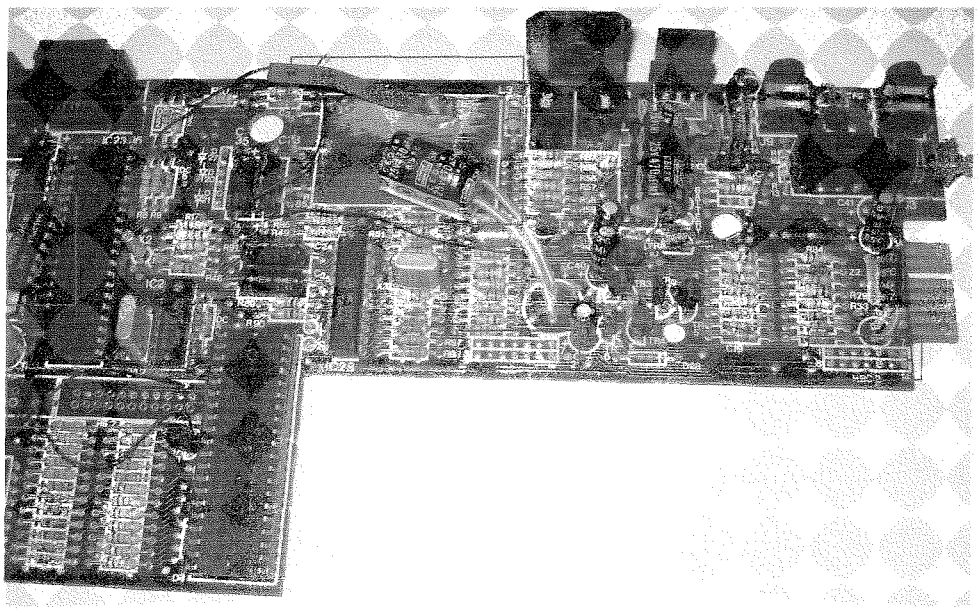
Thanks to my 2009 efforts before the show I was able to present the following lost treasures at the show in my session "Sinclair QL – the Mac that never was...". In this article I shall go into more details than I was able to do in my session.

The (last) existing QL prototype

Picture 1)
This is a pre production QL PCB (motherboard), issue 2, named ZX-83, 3 1983, produced in week 50 of 1983 (8350). I acquired the board from Rich Mellor who got it in the QL lot from Tony Firshman.



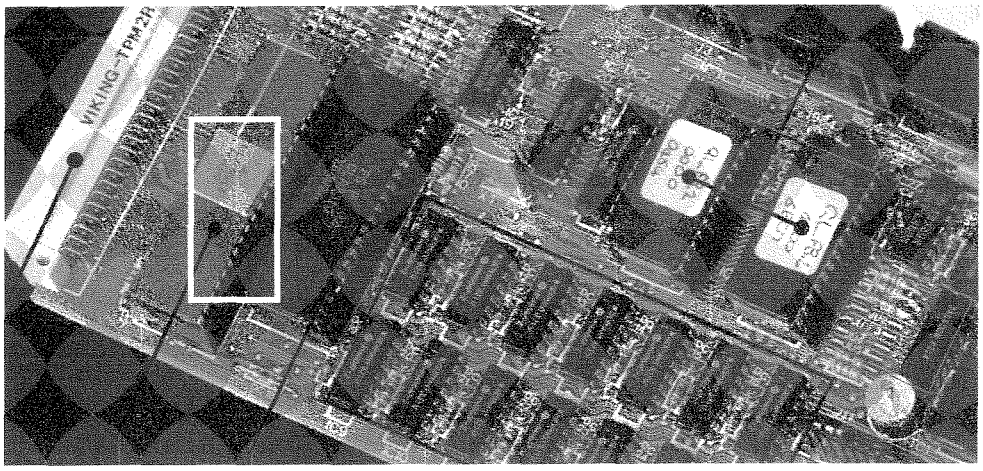
Picture 2)
This board features several modifications and more important many components are not fitted at all.



Referring to Tony Tebbys "QL Firmware Bugs Myth" (QL Today V14I1 and V14I2) this PCB emphasises/ proves the following:

- At least until late 1983 the computer was called the ZX-83 not the QL.
- Physical sizes were frozen at that time, so manufacturing of tooling/case was possible at that time and therefore must not be blamed for the delay in QL production and shipment.
- The board already had two (EP)ROM sockets for the firmware. It is configured for 2x 27128 EPROMs (IC17 added, JU1 and JU6 set, JU2/JU3/JU4/JU5 empty) giving 32Kbyte address space for the firmware (Picture 1).
- A SN74LS04N Hex Inverter was loosely added to the board. What for?
- No TV modulator, no Keyboard and no Microdrive sockets are fitted to this specific board (Picture 1). So this board was not (part of) a full working prototype. But the board seams to be good enough to verify basic function of the chipset, the board and the firmware and maybe use of an external (Spectrum) Microdrive.

The (first) QL shipped

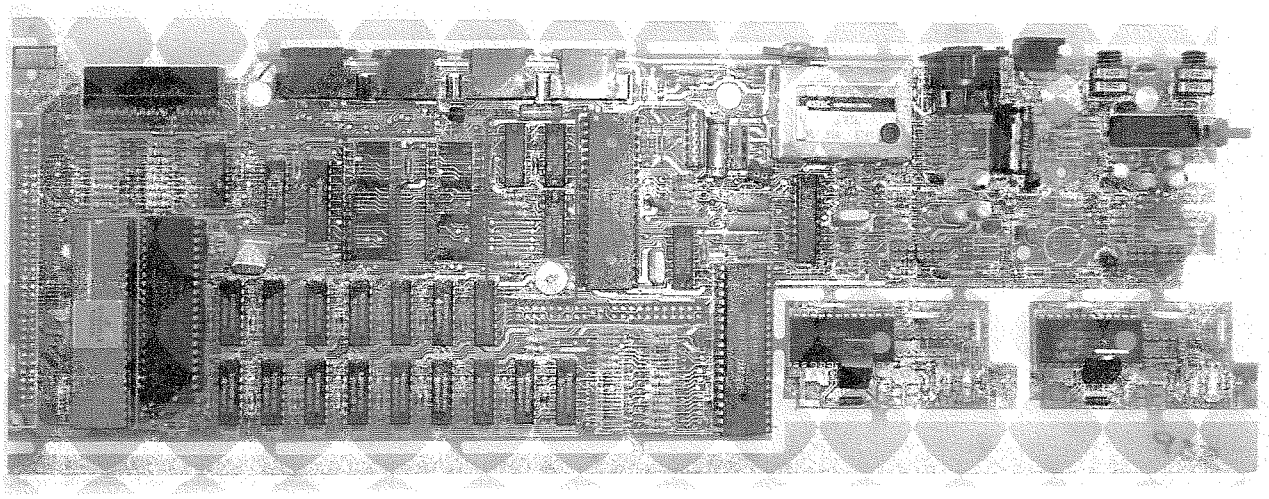


Picture 3)
A hi-res magazine scan showing an early QL PCB, issue 4, named QL, 3 1984. Also this board is configured for 32Kbyte firmware using EPROMs. This was a dongled QL with FB firmware (Qdos v1.00) in 3 EPROMs.

From various magazine reviews of the first QLs shipped to reviewers and first customers at end of April 1984 it shows that the kludged QLs all had issue 4 motherboards.

An early QL with serial number D04-001371 sent by a customer for "firmware update" was sent back to him with a replaced PCB (now issue 5), ROMs fitted and further build-level modifications added, but still with the original socketed chipset (68008, ZX8301/CLA2310, ZX8302, 8049 and the old heatsink.

The (first) production run QL motherboard



Picture 4)
This is a (still) uncut production run QL PCB, issue 5, named QL, - 1984, produced in week 15 of 1984 (8415). Also this board is configured for 32Kbyte firmware using EPROMs. This board has no Expansion-Port and no Keyboard connectors fitted at all. I acquired this board from Rich Mellor who got it in the QL lot from Tony Firshman.

All pre-Issue 6 boards look very similar. They have the same IC population. Issue 6 and 7 differ in terms that they miss TC1 and IC27 but hold IC38 (see white frame in the middle of the picture).

I know of at least one early QL that has an issue 5 PCB which with exactly the same 8415 mark.

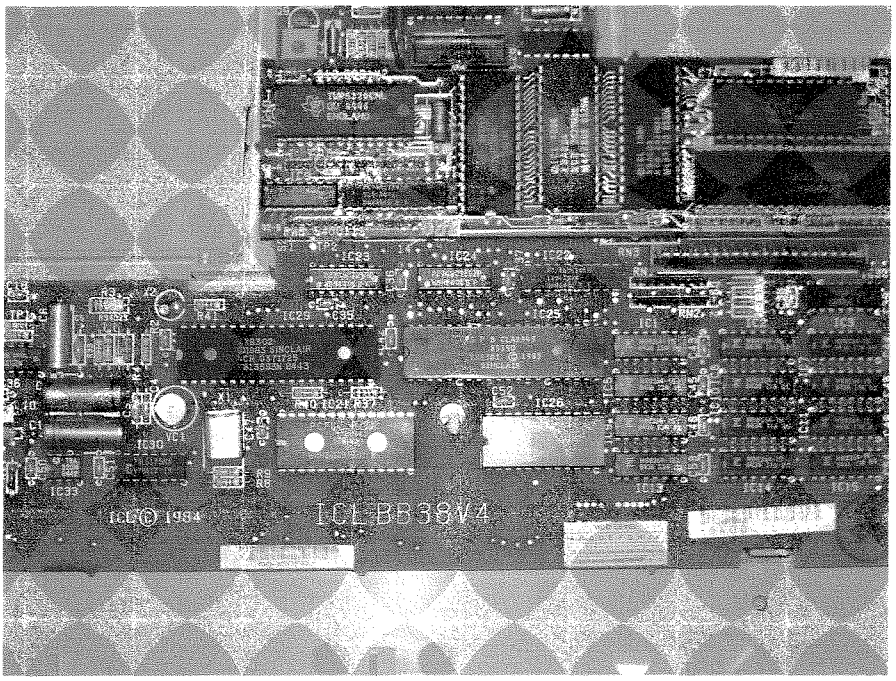
Kludged early QLs (Pre-Issue 5 PCB and ROM-Port Dongle) sent back by the customers for "firmware" upgrade were sent back to them with a PCB issue 5 and "new" firmware fitted.

Referring to Tony Tebbys "QL Firmware Bugs Myth" (QL Today V14I1 and V14I2) this PCB emphasises/ proves the following: **At least in week 14 (starting Monday April 9th 1984) the factory (which one?) produced incomplete populated issue 5 motherboards for whatever reason (Picture 4).**

The third original ZX83 based computer

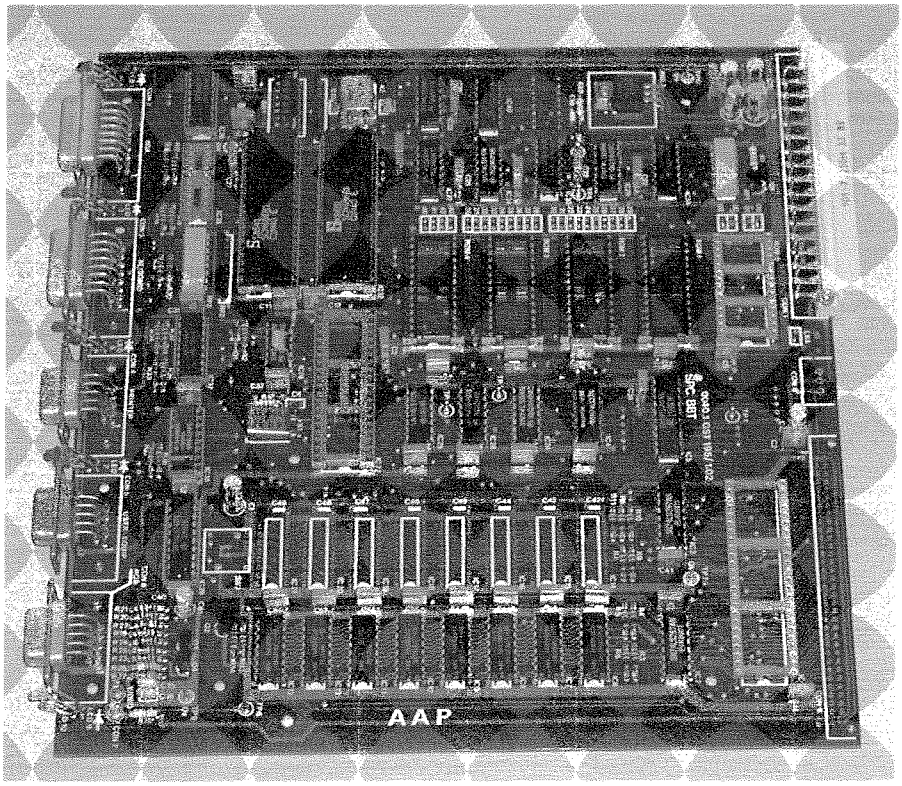
All readers know the **Sinclair QL** (ZX-83) is Sinclair's own computer based on the ZX83 design. Many readers know about the **ICL ONE PER DESK** (OPD for short) computer and the rebranded but technical identical models **BT Merlin** and **Australian Telecom Computerphone**. The OPD was launched at end of 1984 after a three years joint but mismanaged project between ICL and Sinclair.

Picture 5)
Closeup of the ICL OPD motherboard with the ZX83 chipset of the Sinclair QL.



Some later QL compatible computers were based on the Sinclair QL motherboard (Thor, QLT, SPEM, ExeQtor, etc.). All known other and later QL designs like QLT, Thor XVI, Futura, Qx0, etc. were not based on the ZX83 chipset nor the Sinclair QL motherboard. OK, there's Aurora which is a kind of a hybrid between the ZX83 design and some more approached design by Nasta (ZX8301 replacement). But that's not what I mean with the third original ZX83 based computer.

Picture 6)
A single board computer based on the ZX83 design made by GST for Australian Associated Press (AAP for short).
Picture courtesy Tim Ward (former GST employee).



Tim Ward told me the following:
"The other thing I've got is an example of the re-worked QL which we did (under licence from Sinclair) for AAP as a financial information terminal! What I've got is the PCB but minus the processor and the ZX8301 (we didn't use the ZX8302). I can probably find it and let you have a photo if that's of any interest. Bizarre project, but it got me three trips to Australia, including stopovers in Hawaii and India, never had a job like that since I'm afraid."

Furthermore he told me about the board specifics:
"The missing chips are the 68008 processor, the ZX8301, and the PROMs containing the code. Also missing is the rack mounting hardware and front panel (goes along the edge with the connectors and LEDs, obviously). IIRC the fifth socket, in line with the four PROM sockets, just brought out the address lines - what you did was plug in a little daughter board with some LEDs on it, giving a binary read-out of the address bus for debugging. (So, if the soak test code failed[#], it would indicate the particular error code by looping at a particular address which could be read off from the LEDs.)

The rear connector is just power.

The front connectors are keyboard, monitor, and some serial ports for data.

The device ran 68K/OS with, in due course, AAP's application software, which I never saw.

The boards were rack mounted - we had to demonstrate that the board would drive a monitor via 100m of co-ax. So the idea was to have a rack of them somewhere then cables to the keyboards and screens at the dealing desks. (To be sure we met this requirement, after the electronic calculations and design were done we bought 200m of co-ax and spread it out round the car park at Willingham, with the board and monitor in different buildings.)

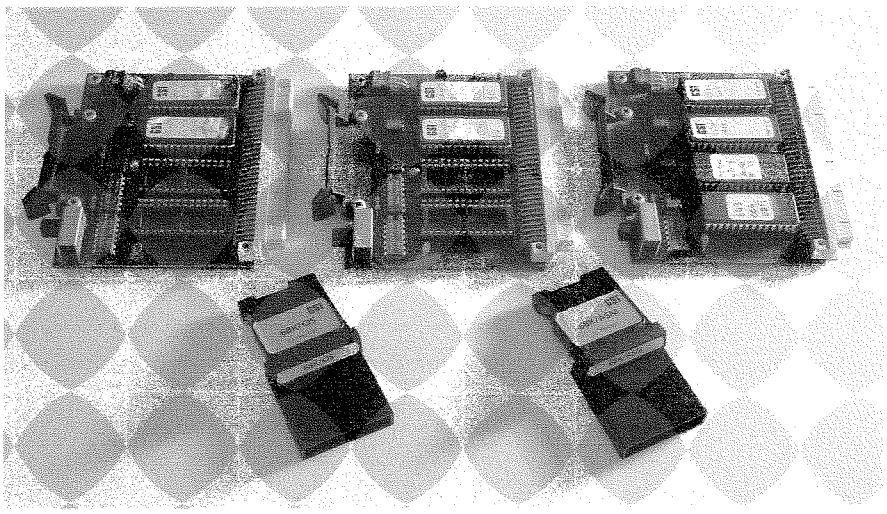
We sold AAP a couple of hundred boards, so they put a non-trivial amount of money into it, including paying for me to spend a month in Sydney hand-holding their development. What I don't know is whether the devices were ever actually used in the end.

[#] Oh yes, the soak test. The contract said the boards had to run for 48 hours. However they routinely failed at something like 63 hours. So AAP asked us nicely if we happened to know why, they appreciated that they couldn't insist on us making it work longer than the 48 hours in the contract. Turned out to be a counter overflow in the test software."

The original OS

As most readers know GST was commissioned by Sinclair to write the OS for the ZX83. Sinclair selected Tony Tebby's in-house design Domesdos to become Qdos. Talking on Sinclair, ZX83, the QL, GST and their 68K/OS Tim told me:
"The press launch [of the QL], at the Intercontinental on Park Lane. I was the one on stage pressing the buttons for the 68K/OS demo. I wasn't really in a fit state to do this, on account of the dinner the night before - it was only after I'd started on the third G&T that I spotted the little notice that they only sold doubles in that bar, and some of the party insisted on buying lots of decent, expensive, wine and port - and also the Bucks Fizz for breakfast didn't help much.

The demo of the graphics was a Christmas scene that Howard Chalkley had put together, complete with snow falling outside the window. As the launch was well after Christmas this was a bit of a clue that the launch had got delayed a bit beyond the original plan!"



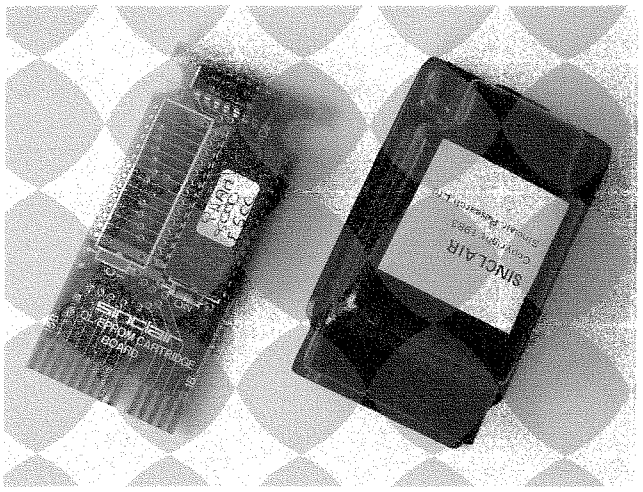
Picture 7)
A collection of three GST 68K/OS OS-Switchboards for the QL. FLTR: Dilwyn Jones' board (now owned by Marcel Kilgus, kindly lent to me), Jeff Fenton's board, John Bradley's (ex GST employee) board, an original set of Microdrive cartridges as shipped with the board. I acquired the cartridges from Rich Mellor, the two boards from Jeff Fenton (GST founder) and Paul Tankard. 68K/OS consists of two 27128 EPROMs making 32Kbyte. The board on the right, equipped with four EPROMs (called Demo-Board), hold the OS and a 32Kbyte ROM-Disk with loadable software for it (you didn't need the Microdrive cartridge then to demo some 68K/OS programs).

Referring to Tony Tebby's "QL Firmware Bugs Myth" (QL Today V14I1 and V14I2) the "Sinclair used 68K/OS as Qdos to demo the QL at launch" story emphasises/proves the following: **At launch prototype QL(s) were equipped with two 27128 EPROMs containing GSTs 68K/OS.**

Thanks to the help of Danielle Terdina it is possible to run 68K/OS on Q-emuLator. Currently only the beta version 2.9a1 of Q-emuLator can cope with the 68K/OS image.

Picture 8)
68K/OS running on Q-emuLator.

The dongle containing good firmware



Picture 9)
Original (unmodified) dongle with AH (Qdos 1.02) firmware.

Referring to Tony Tebbys "QL Firmware Bugs Myth" (QL Today V14I1 and V14I2) this dongle emphasises/proves the following: **Not all dongled QLs had pre-test firmware (FB/Qdos 1.00, PM/Qdos 1.01). At least some QLs were equipped with good working firmware (AH/Qdos 1.02).**

GMOVE, not Gspot

As it is well known, Linus Torvalds used and programmed a Sinclair QL before he created what became Linux. While hacking on the QL, Linus wrote quite a few programs for it. For many years none of them seemed to have survived. After a huge research I'm very proud to present at least one piece of QL software written by Linus Torvalds.

LINUS TORVALDS

GMOVE

●● Onko joskus tullut eteen tilanne, jossa olisit halunnut siirtää osan ikkunasta toiseen paikkaan? Homma ei onnistu ilman oheista Basic-laajennusta. Se lisää QL:n käskyvalikoimaan GMOVE-käskyn, jolla voi siirtää grafiikkaa esimerkiksi piirto-ohjelmissa.

GMOVE (#ch),x1,y1,l,k TO x2,y2

out of memory —

out of range —

Ohjelman käyttö

mdvl_GMOVE.A

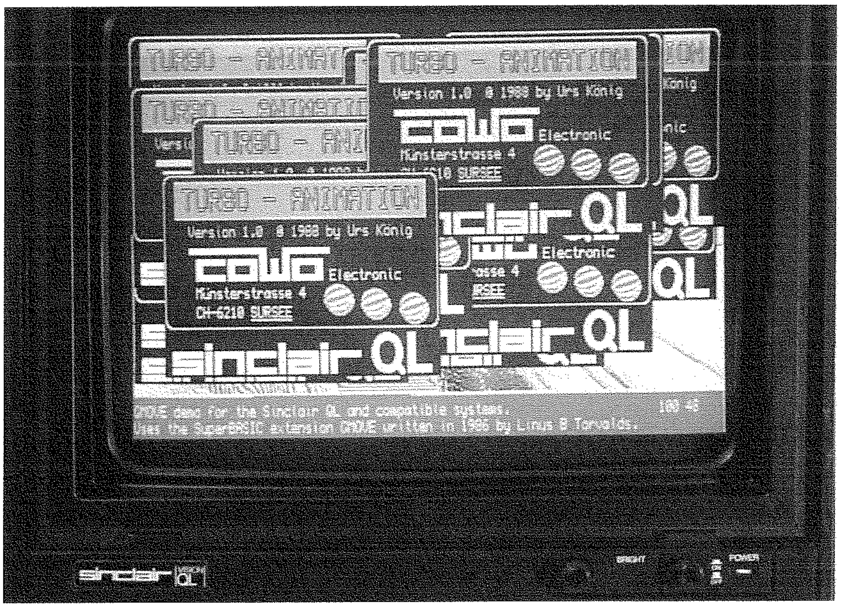
A=RESPR(1000):LBYTES mdvl_GMOVE.A

CALL A

Sinclair QL

BITTI 11/86 63

Pic 10) Scan of the GMOVE article with a hex loader listing as appeared in the Finnish computer magazine MIKRO BITTI in November 1986.



Picture 11) A small demo written in SuperBASIC which makes usage of the GMOVE extension running on a standard QL.

References/External Links

I have not added type-in links/URLs to this article. But for each topic there are specific infos, links and downloads on the recently updated "QL & Mac are 25" and "QL is 25" webpages.

http://tinyurl.com/ql-mac-show
http://tinyurl.com/ql-is-25

If you're interested in more audio/visual QL resources (including the pictures of this article) you my try out one of these URLs:

http://tinyurl.com/ql-story
http://tinyurl.com/ql-pics
http://tinyurl.com/ql-videos

Desperately Seeking QL

Although I made good progress in completing my QL collection last year I'm still looking for the following:

- Video footage on the Sinclair/QL in TV programs (on VHS tape or as files).
- Photographs of Sinclair/QL appearance in IT-shows such as Personal Computer World show (PCW), London, Which Computer? show, Birmingham, Earl's Court Computer Fair, London or ZX Microfair, London (1984 to 1988).
- More software for GST's 68K/OS beside the two bundled Microdrive cartridges.
- More QL software/articles written by Linus Torvalds.
- Sinclair QL FB ROM (physical EPROMs or binary file).
- Sinclair QL PM ROM (physical EPROMs or binary file).
- Any Sinclair QL Professional Computer, serial number (S/N) pre D04-001371.
- Any Sinclair QL Professional Computer, S/N post D16-122418, a D17 would be very nice.
- Any Sinclair QL Professional Computer US-Edition, S/N post S13-005854.
- Any Sinclair QL Professional Computer German-Edition, S/N post SG18-010800.
- Other Sinclair QL regional editions (Spanish, French, Italian, Danish, Turkish, Greek, Portuguese, Norwegian, Swedish, Finnish and Arabic).
- Any names or contact to people who worked at a subcontractor of Sinclair Research like Thorn (EMI) Datatech.
- QUEST CP/M 68K for the QL.

Just drop me a line (mailto:**urs_koenig@bluewin.ch**). Thank you very much.

I hope you enjoyed this journey back in time. QL forever!

25 Years - Part 2

by Tony Tebby

Welcome to the next part of our series. More has been prepared, and Tony writes "Just the FUTURE to do". We ended last issue with "25 Years - out of the Morrass ..." and continue:

25 Years - . . . into oblivion

Domesdos hits the fan

The peer review

How well was the system received in the computer science world? Not very well. The main criticisms were that it did not incorporate all the idiocies that I had deliberately avoided. In other words, it was criticised for how it worked rather than whether it worked well.

Other criticisms centred on the OS interface. "It stinks" was one verbal reaction - the silent reactions were probably worse. For me, the oddest thing about the criticisms was the irrationality - the thing that stank the worst seemed to be the I/O calls. These were designed to meet a wide range of requirements including communications and interactive use where an application might have to react if data was not received within a given time. This meant that all I/O calls had a timeout. This combination of scheduling (for the timeout) and data transfer seemed to be fundamentally offensive to any true believers in operating system purity. Apparently the "correct" way of dealing with this was to make the application spin in a tight loop checking the elapsed time, checking for data, transferring the data and processing it a byte at a time. Is this really better, in any respect, than making a single operating system call? A lot of people thought so.

The third group of criticisms centred on the security or, rather, lack of security in Domesdos. The main concern was that as the hardware had no protection mechanisms, the system did not try to provide a false sense of security by implementing arbitrary, highly obstructive protection mechanisms that would, in any case, have been totally ineffective on the QL platform.

Users' reactions

I have no doubt at all that there were many users who were disappointed in the system and there really was no adequate documentation on the operation of the system, no guidelines and interface information. On the other hand, "ordinary" computer enthusiasts did not suffer from the preconceptions of systems "experts" and so a fair number succeeded in making the system do all sorts of extraordinary things. The lack of formal guidelines for programming showed up some unforeseen characteristics: on the one hand, the ability of the system to put up with extreme abuse while continuing to function and, on the other hand, the ability of a certain type of person to take great pleasure in exploiting all the undefined holes in the system interface that will always occur if you adopt the "garbage in, garbage out" approach used by Domesdos.

My own assessment

Relief that it worked. Although I had believed the theoretical basis to be sound, it was still a relief that I had not completely screwed up.

Anger at myself for having compromised the integrity of the system to incorporate SuperBASIC rather than changing SuperBASIC to be "Domesdos friendly".

Pleasure when, on a simple system test with 100 application programs, the QL outperformed a VAX running VMS.

Frustration that I could not redesign it from scratch and "do it right" the second time around on a better platform without the compromises for CGA compatibility and the QL hardware.

Meeting the design criteria

Compactness

The first version of Domesdos met its target for compactness. The core occupied less than 5 kbytes even though the range of core operating system functions was very much more comprehensive than Unix. The complete set of device and filing system drivers occupied less than 10 kbytes.

The compactness of the core functions was largely due to the use of pseudo code for programming, the "real world" approach to task management, the use of events in place of synchronisation and the adoption of regular, coherent interfaces.

The compactness of the device and filing system drivers was largely due to the simplicity of the interfaces provided by the data design concept and the use of intrinsically safe intertask communications between the interrupt servers and the operating system functions.

The compactness was not achieved by "crafting" the code. Subsequent development indicated that some sections of the code could have been reduced by more than 20% by more careful coding.

Efficiency

The first version of Domesdos was not as efficient as it could have been, but all the targets were met. Basic operating system functions were more than an order of magnitude faster than Unix SV. One hundred applications running simultaneously did not bring the system to its knees. The only real bottleneck in performance was the display handling, but this was not a basic design fault, it was simply an unsatisfactory trade-off between speed and size of code in the first cut that was never to be revised.

The efficiency came from an awareness of the costs of various options that were selected and the many options rejected, the absence of any intrusive synchronisation mechanisms, the use of a real memory address space model and the extensive use of shared data structures that this memory model and the intrinsically safe access mechanisms made possible.

Once again, the efficiency was not obtained by crafting the code. Some critical sections could have coded up to 30% faster.

Reliability

The reliability of the system has three different aspects. There may be problems with the fundamental system design, there may be problems recovering from or handling external errors (hardware faults, exception conditions, OS call parameter errors, etc.) and there are simple coding errors.

These three sources of errors were all dealt with by design.

The probability of simple coding errors was reduced by using a data design approach translated into machine code via state diagrams and pseudo code. The error handling was made simpler by having a single, rather primitive, mechanism for reporting errors from procedures which, together with the regularity of the interfaces, reduced the probability of errors in the error handling. The two major differences between Domesdos and conventional operating systems were, however, the total elimination of synchronisation and mutual exclusion mechanisms and the regularisation of the operating systems interface which, between them, eliminated the root causes of most of the known design problems in operating systems at the time.

In a fit of hubris over this "design for reliability" approach, the new operating system was called Domesdos (a home (domestic) DOS, even though it was designed for business use and it was a ROM operating system and not a disk operating system) after the slogan for a brand of bleach "Domestos" which "kills 99% of all known germs (bugs)".

How well did this approach succeed in eliminating bugs at source? In reality better than the system's reputation might indicate. When the first QLs were shipped, Sinclair set out to create the impression of bug-ridden software to mask the chronic hardware problems. To back this up, machines were deliberately delivered with pre-test firmware (c.f. "QL firmware bugs myth"4). Furthermore, Sinclair's failure to organise any form of operating system documentation led to a whole bug-hunting industry, with journalists fighting it out to produce the most extravagant lists of "bugs" without actually knowing what the system was meant to do. Mark Knight gathered all these together, weeded them and created the "definitive" list of 77 bugs and quirks, most of which were only in pre-release versions or the SuperBASIC interpreter and its associated utilities, procedures and graphics. These bugs are analysed in "QL ROM bugs - an annotated list"5 which shows that there were 10 bugs and serious quirks in the first release version (V1.03 JM) of the operating system and device drivers, excluding SuperBASIC and the graphics (see Box 3).

It is quite possible that there were other errors and the system could have been better, but by industry standards, for an operating system developed from scratch within a six month deadline using new, radically different, paradigms, it was probably better than industry average.

Predictability

From the a user's point of view the ratio of the median response (typical response) to the worst case and the incidence rate of long delays are two reasonable measures of the predictability.

In some respects, users would have found the system occasionally slow and unpredictable. With a mean Microdrive access time of 3.5 seconds, fetching data from files was bound to be slow. By comparison with simple buffering, the file imaging (borrowed from CST's 68KOS) and pre-fetch strategies significantly reduced the median access times but could do nothing to improve the worst case delays, thus the predictability measured as the ratio of median to worst case delay was made worse. These strategies did, however, significantly reduce the incidence rate of long delays .

On the other hand, the character drawing speed was certainly too predictable! The screen driver was designed for compactness first, speed was only a secondary consideration. The most compact design was to draw all characters using the same code, regardless of the complexity of the operation. As common cases could have been be drawn much quicker than the general case, using separate code for these would have increased the apparent performance at the cost of predictability. It should have been done.

Accessibility

Domesdos was very accessible.

The end point

The release of the QL was also the effective endpoint of development of that operating system concept. The general hostility of the computer science world to the concept, coupled with the fact that the system was closely tied to a flawed hardware platform and Sinclair's decision to make the firmware a scapegoat for the early QL production problems would have turned Domesdos into a little footnote in the history of operating systems if the Sinclair marketing department had not buried it.

4 http://www.t-t-web.com/OS/QL_firmware_bugs_myth.pdf

5 http://www.t-t-web.com/OS/QL_ROM_bugs_list.pdf

Domesdos was now called QDOS by the marketing department. Possibly they did not know that QDOS already existed and was alive, if not very well, and that Bill Gates had been to Sinclair Research in a attempt to license it to Sinclair for the QL. Possibly they thought that the operating system on the QL had been licensed from Microsoft! As the ultimate insult, they took its name away, and hung the albatross-like "QUICK and DIRTY OS" epithet round its neck.

Box 3 - First release bug list

The numbers are those in Mark Knight's original "definitive" bug list

1. A trivial coding error (bug 57) checks only one plug in card (Plug 'n Play in 1984). Simple workaround.
2. Serious oversight (bug 8) remapping the colours on changing from eight colour to four colour mode. The driver did not check whether the "ink" and "paper" mapped onto the same colour in the 4 colour mode.
3. Simple coding error (bug 15) panning a window less than 8 pixels wide. Why would you do this?
4. Trivial coding error (bug 43) filling a full screen width block. It actually did nothing at all, I do not know why.
5. One bit coding error (bug 37) closing a serial port. A little endian IPC connected to a big endian MC68000.
6. Serious oversight (bug 33) pre-fetching Microdrive sectors when you had run out of memory. This rather reduced the buffering efficiency.
7. Serious coding error (bug 19) in the Microdrive device driver opening MDV8. There was no MDV8!
8. Fatal oversight (bug 71) sending a null file over the network. One end just waited for nothing for ever.
9. Serious coding error (bug 50) in the Trap #4 / Trap #3 patch to cater for a "moving task" (SuperBASIC).
10. Design error (bug 49) handling job release events. This did not allow for nested release events making it possible to break the system using apparently legitimate OS calls.

25 Years - Development of the Domesdos concepts

SMS2

When it had been well established that Domesdos did work reliably and significantly more efficiently than conventional multitasking systems, I set about writing version 2. The interface was cleaned up to remove "QL nasties", which meant that there was no SuperBASIC, graphics or Microdrive support and the OS calls that had been patched in to support direct access to the QL hardware were not included. This SMS2 (Small Microcomputer System V2) was hardly any larger than the original Domesdos core and, typically, important operating system calls were 30% to 100% faster as register handling was optimised for the more complex calls, rather than the simplest, and the code was crafted more carefully.

There was no permanent user interface program but, whenever there were no jobs in the system, a default application (job 0) was started. The only job 0 application that was written was a simple command line interpreter that could read from a file or the console. This was, effectively, going back to the original Domesdos concept.

This system was implemented on the Atari ST monochrome system and feasibility trials were carried out on a small number of embedded systems. It was never made available commercially on "standard" platforms and no project using it ever made it to market.

Windowing software

Windowing is not normally a fundamental part of an operating system, it may be a function of the display driver or it may managed by an independent task, in either case the emergent windowing systems 25 years ago all relied on the applications programs doing most of the work, although some of this work may have been hidden in "API" libraries. This approach of requiring application programs to re-draw, at any time, regardless of what they were doing at that time, any parts of their windows that became uncovered, was so costly and fragile that it seemed that the real way forward was hardware windowing. This should have been able to provide a much better price/performance ratio than a 100% software approach.

The original Domesdos display driver was based on writing to windows rather than the whole display. This was a prerequisite for a extension to a windowing system although it did not actually provide windowing facilities. The upgrade path was laid out but no more than that.

The QJump Pointer Interface QPTR was designed as a stopgap measure for the QL. It provided true windowing without clipping, it was "optimised" for a platform three times slower than the recently released Apple Mac, it had to be compatible with existing QL software and it was based on the assumption that windowing hardware would soon become available. As a result, the overlapping window pro-

blem was dealt with by freezing partially covered windows: a less than satisfactory approach, but, possibly, the only practical short term solution.

At the "presentation level" (the Window Manager), however, there were a number of innovations which were intended to deal with some of the less desirable features of what was to become the "standard" windowing interface as presented on the Apple Mac, the Commodore Amiga and the Atari ST. Some of these innovations (right click menu, button bar, etc.) were adopted in improved form in later mainstream systems (context menu, task bar / dock, etc.). Others, such as the Hotkeys, were not.

SMSQ

SMSQ was SMS2 to which SuperBASIC compatibility was retrofitted. It was intended to provide a QL compatible operating system for various QL emulators. In general it was much faster than QDOS (executing in equivalent speed memory). The expanded console driver could draw strings of characters at speeds within a few percent of the QDOS add-on record holder and it could draw single characters faster. It was purely retrospective and did not further the Domesdos style OS principles.

SMSQ-E

SMSQ-E was SMSQ bundled with the standard QJUMP Extended Environment. There was no development effort available to provide a mechanism for writing to buried windows which had become practical with the availability of "QL compatible" platforms 5-10 times faster than the original QL. Various enthusiasts and third party developers produced their own solutions by continuously updating the display buffer from the off-screen window buffers. This was a bit of a patch that was later adopted by Mac OS X and Windows Vista.

Minerva

Minerva was a re-engineered QDOS / SuperBASIC ROM destined only for retrofitting to QLs. it provided improved performance with a reasonably high level of QL compatibility. Although it provided interpreted multitasking BASIC it did not represent any form of advance in operating system principles.

Windowing hardware

A hardware windowing display circuit was designed using 1980s technology. The principle was very simple: for each frame there was a table of "pixel runs": the start address in the display memory, the number of pixels in the run and the attributes (colour depth, colour map, and display/porch/blanking/sync). While this made displaying a line slightly more complex than a simple rectangular memory map for the whole screen, it greatly simplified the generation of the blanking and synchronisation which were just special pixel runs and entirely defined in the pixel run table. The principal additional cost component was a deep FIFO to smooth out the pixel rate at the window edges.

This system allowed for arbitrary shaped windows, which could be moved simply by changing the pixel run table. At the time, large display memories were expensive so the design allowed different windows to use different palettes or different colour depths to economise on memory usage.

It never made it to market.

Stella

Stella was designed as a "lets get it right the second time round" Domesdos. After a few years, I felt that I understood most of the problems with the original Domesdos. Stella was based on an updated, more rigorous version of the Domesdos design principles applied to a wider range of platforms.

- Elimination of the UNIX legacy
- Extended data design concept
- Wider OS interface base
- Higher efficiency
- Less memory constrained
- Generalised entity management
- External event management for guaranteed real time response
- Rationalised and more rigorous ownership and usership concepts
- Mix and match modularisation of core operating system functions

Stella on test

At the request of someone at Sun Microsystems, Stella was benchmarked against Unix SVR4 (Solaris 2) on a Sun3x equivalent platform (Sun's project to replace BSD Unix was not giving the desired improvement in performance). The benchmark conditions were not very well defined but, on the first tests, Stella outperformed SVR4 by around 2 orders of magnitude on simple OS calls and filing system operations. My contact in Sun told me that he could not possibly pass these figures onto his project manager as he would lose all credibility!

Stella on the market

Stella was implemented as the embedded operating system for a number of projects, but none ever made it to market.

25 Years - 1984 to 1986 - GUIs and RISC

The first mass market GUI machines

Apple Macintosh

The Apple Mac was launched soon after the QL, although it had been in development for nearly 5 years. With a price tag three times that of the QL and operating system development costs about 50 times that of Domesdos, the machine was far more polished than the QL. It has since become a landmark, for many of the wrong reasons. Many histories of the development of personal computers placed the Apple Macintosh as the computer that started the windowing revolution. In reality it had very little lasting effect.

A recent history of the Mac states "it had very little memory . . . low processor speed and limited graphics ability". The "little memory" was surprising - it had no more memory than the vastly cheaper QL and less than typical for a new PC but was aimed above the PC market. Given the lack of expansion capability, this was a clear marketing error. The "low processor speed" is more interesting. The processor was comparable to a PC of the period, but the system was slowed down by the windowing software which was based on the principle of pushing all the real work up into the application. It is not that the software was particularly inefficient, it was more that the windowing principles were totally inappropriate for the single tasking Mac.

The operating system was reasonably compact, having a basic I/O and filing system, primitive memory management, scaled pixel fonts and desktop in 64k RAM.

It fell down very badly on accessibility. Early software development was limited to a few privileged partners who had received advance specifications and cross development systems. Ordinary developers had to wait two years before the MPW native development platform became available and they were able to sample the dubious delights of programming in an environment where the operating system imposed very serious constraints but provided very little assistance.

This lack of system accessibility meant that, despite its popularity with journalists, the persistently over-the-top reviews and an extravagant marketing campaign, the Mac did not manage to stop the much more basic MSDOS PCs becoming dominant. Software development was just too difficult.

Over the next few years, Apple's marketing strategy for the Mac was pursued through advertising and vigorous, totally baseless, legal action presenting Apple as the underdog suffering under the market dominance of IBM and from theft of its intellectual property. At the time the campaign started the IBM PC was a long way behind the Apple II in terms of installed base and third party software support - IBM was the underdog - and the disputed intellectual property did not even belong to Apple.

Although both parts of Apple's strategy were based on misrepresentation, this has left a lasting impression that the Mac was an innovative machine that was pushed out of the market by IBM monopolistic marking practices. In reality, its software was 99% derivative and it was unable to compete with Apple's own Apple II series.

The Mac had one feature borrowed from the Lisa that did get copied. This was the menu bar at the top of the screen. The menus for each program or window were not part of the program's windows, but "belonged" to the screen as a whole. This bizarre arrangement also appeared on two computers that set out to succeed where Apple had failed: the Atari ST and the Commodore Amiga.

The Atari / Commodore twins

The Commodore Amiga (which was actually an Atari games console) and the Atari ST (which was designed by Commodore or by engineers that had left Commodore, depending on who you believe) both had user interfaces based on the Mac. Put on the market a year after the Mac, both had more powerful hardware than the Mac and both were milestones in the development of personal computer system software.

The similarities between them and the differences between them and the Mac were striking.

- Their MacAlike GUIs were patched in at a late development stage, whereas the Mac was designed that way.
- They both opted for the efficiency and simplicity of mono-spaced fonts, whereas the Mac was designed to use proportional, scalable, fonts.
- They were both designed for expansion whereas the Mac was a take-it-or-leave-it closed system.
- Their systems were jumbled heaps of recycled bits of archaic systems and reverse engineered clone software, whereas the Mac system was designed for the job.

It is this last aspect that made them the true precursors of the systems that have dominated systems development over the last 25 years.

There were, however, significant differences between the operating systems.

The Atari ST, with its rather simpler system, easily outperformed the Mac although its inbuilt mono-spaced text made the screen (desktop) look look rather clunky. A large part of TOS (The Operating System) was recycled from CP/M.

The Amiga screen (workbench) also looked rather clunky. However, apart from the demos which bypassed the operating system, it was slow (particularly the filing system), even by comparison with the Mac, despite the Amiga's more powerful hardware. Moreover, the although the operating system was arguably less capable than the Mac (the Domesdos experience had shown that scalable fonts "cost" far more than multitasking), the operating system was grossly oversized, eating up about 4 times as much memory as the Mac. Part of the reason for both the low performance and the enormous size of the operating system was that it was bodged together from three separate and largely incompatible units: the recycled Exec and AmigaDOS and the reverse engineered MacAlike Workbench.

What was amazing was that the press, as a whole, accepted the proposition that the Amiga was slow because it had a "powerful" operating system. This gave a whole new, hitherto unsuspected meaning to the word "powerful".

The Amiga operating system was "powerful" in the sense that, by comparison with other personal computer operating systems of the period, it consumed vast resources and, therefore, required a powerful hardware platform for it to work at all.

It is difficult to see "powerful" in this sense being applied to any other engineering domain. Can you imagine automobile journalists raving over the "power" of a car just because it guzzled fuel in unprecedented quantities despite an acceleration that would make model T Ford blush and a top speed that guaranteed that you could never never pick up a fine?

24 years on, "powerful" in the sense "memory hungry and inefficient" is now accepted usage. Only today, I was helping out someone who was lamenting that the latest software versions (and you must have the latest!) were too "powerful" for his 2 year old machine.

The Amiga, therefore, was the computer that established

- "scrapyard" (recycled, re-used) software technology,
- "bloatware" and
- the new meaning of "powerful".

1984 - X Window System

The X Window System is now closely associated with Unix but the original version "W" was written for the V operating system. The development of X at MIT, which started a year after Domesdos, has had a long lasting impact in two respects: not only is X the dominant display management software in the Unix world, but the egocentric minimalist design philosophy adopted for X has also been recycled many times in computer science. To be fair to Bob Scheifler and Jim Gettys, the developers of the X Window System, X was never designed for the purposes for which it is now misused and they did not originate the egocentric minimalist design philosophy: they just gave it an air of respectability by formalising it as part of their 7 golden rules.

Box 4 - X Window System's 7 golden rules

1 Do not add new functionality unless an implementor cannot complete a real application without it.

If this golden rule were to be believed and followed rigidly by systems developers, it would be a disaster for the world. This is the minimalist philosophy in its most rigid form. As "application implementors" can, in principle, do anything that an operating system implementor can do, this implies that the system should do nothing.

With this rule, there is no question of whether implementing useful but non-essential facilities could be better for the overall system (operating system + applications). All functionality is dumped on the applications developers regardless of the overall cost in reliability, performance and development effort.

This rule, although being the first, most important rule, is so stupid that it was not was not followed by the developers of the X Window System. This provided a number of non-essential drawing primitives, in particular glyphs (characters). Drawing glyphs is clearly not "required" as applications can, of course, do it themselves.

2 It is as important to decide what a system is not as to decide what it is. Do not serve all the world's needs; rather, make the system extensible so that additional needs can be met in an upwardly compatible fashion.

There are two different completely different concepts in this golden rule. The first is a platitude that restates a fundamental engineering rule "design to purpose". This implies that the purpose is known - for operating systems that means reading the future. The second would seem to be good advice, but is usually taken to mean **"do not do today what you can leave until someone else is forced to do it".**

3 The only thing worse than generalizing from one example is generalizing from no examples at all.

This was prophetic. Although originally the X Window System avoided generalisation by being designed for a single case, it must be the most significant example of "generalising from one example". Designed for a system architecture that was already archaic (X Terminals connected to a centralised time sharing system) it has been generalised for architectures for which it is fundamentally unsuitable (such as stand-alone workstations).

But there is much worse than the X Window System. The 1960s operating system theories that have been the cause of so many system design errors and problems were based on a, rather ill defined, abstract system architecture that corresponded to no real computer. They are the prime example of "generalizing from no examples at all" so maybe this golden rule is right.

But are there other things worse than generalising from one example? Unix generalized the "file" concept to apply to I/O devices, with horrific results. MSDOS had no generalized I/O at all. If you believe that the MSDOS approach was worse than Unix (I am not taking sides) then you will have to take it that not generalizing at all is also worse than generalizing from one example.

4 If a problem is not completely understood, it is probably best to provide no solution at all.

If you cannot completely understand the problems that you have to deal with, first try changing the problem, if that does not help, change jobs to something less mentally exacting (a trader in the derivatives market, for example). Copping out by providing no solution at all is, at best, lazy and irresponsible, at worst, criminally negligent.

5 If you can get 90 percent of the desired effect for 10 percent of the work, use the simpler solution.

Even if 90% were good enough (would you buy a car where only 90% worked?), it is highly improbable that you could make a system 90% functional for only 10% of the effort. This is no more than a justification of **"if you can get 10 percent of the desired effect without thinking about it, that's OK, take a break".**

6 Isolate complexity as much as possible.

The X Window System does some fairly complex things, but they are certainly not isolated into easily maintainable modules, they are built into the amorphous mass. Isolate, in this context should, therefore, be take to mean bury out of sight.

7 Provide mechanism rather than policy. In particular, place user interface policy in the clients' hands.

This is abdication of responsibility. Policy is far more difficult than mechanism. It is very wise to separate policy from mechanism. It is very wise to provide flexible policies. It is very wise to allow to policy to be adapted to changing circumstances. A system with every application imposing its own policies is, however, an end user's nightmare.

X Window System's 7 golden rules

A windowing system has the same requirements for compactness, efficiency, reliability, predictability and accessibility as any other system software, even if "accessibility" for developers (coherence and richness of the functionality) and accessibility for users (ease of use) are measured in different ways. Bob Scheifler and Jim Gettys' seven golden rules, however, do not address any user requirements (neither end users nor developers): they are concerned merely with producing something that gives an appearance of "working" while requiring a minimum of effort.

A fuller analysis of these rules is to found in Box 4, but I have tried to translate these into plain English.

1. Push all functionality possible up to the applications regardless of the effect on overall system reliability, complexity, efficiency or cost.
2. Leave anything that you do not feel like dealing with for someone else to deal with later.
3. Design for specific cases only.
4. Do not do anything that that might require thought.
5. Quick hacks are always justified.
6. Bury your hacks so that no-one will ever find them.
7. Do not bother about what the system will do, just how it will do it.

X Window System's implementation

The early X Window System implementations crawled out slowly until X11 was released in 1988. The system was nowhere near as bad as it would have been if the rules had been followed as it included a large number of useful but non-essential features (breaking the first three rules). It had, however, three major failings: the speed, the complexity required in the applications and the bizarre architecture.

The speed

There seems to be very little concrete information about the speed, apart from endless nagging. In 1992[6], 8 years after the development started, The professor of electrical engineering at Berkeley replied to criticism "with the 50 MIPs computers we are seeing now, performance is not a problem". This reply implies that the performance was a problem up to then and there is much anecdotal evidence to to suggest that the performance remained a problem for most of the next decade. In 2004, FBUI[7] was patched into Linux 2.6.9 to replace X "which can be an impossible burden".

The complexity of the applications

The display refresh policy required each application to redraw its windows when they were uncovered by another window being moved or removed. Although an attempt was made to justify this window redraw policy in a seminal paper published in ACM Transactions, the "justification" is very thin: In reality the policy used by earlier windowing systems was simply adopted without any real consideration of the possibilities for "local display refresh" where the windowing system itself deals with changes in the window arrangement. The basis was that applications were better able to redraw the display than the windowing system itself.

Those who have never tried to program for a mainstream windowing environment may not understand just how far removed from reality this is. This "better" refresh policy requires applications to be able to respond to redraw events within the human perception time (0.1 sec), whatever they might be busy doing at the time, and maintain their own data structures *always* in a state such that the window can be redrawn. Failure maintain the state of the data structures can result, at worst, in crashes and, at best, in the famous "bits of window" left over syndrome. Failure to respond can produce the trails of windows immortalised in the title sequence of the "IT Crowd", or, on more recent versions, swathes of blank screen.

The summary elimination of local display refresh was short termism in extreme. By the time that there were machines fast enough to handle the X Window System, memory sizes had increased far faster than processing power and removed the only real objection to using local display refresh. This made the redraw policy used by the X Window System obsolete and ripe for sending to the museum of horrors.

The oddest feature of this was that X11 did have off-screen window buffers (for flicker-free window updates). It, therefore, had all the code required to write to off-screen window buffers and to refresh the display from these buffers. It just did not do it!

The bizarre architecture

The bizarre architecture of X was more the result of trying to patch the system into computers with inflexible, totally inadequate operating systems than of deliberately bad design. Moreover, as it was

6 I cannot find the reference. It might have been 1993. I cannot find the name of the professor.

7 http://home.comcast.net/~fbui/

designed for multi-vendor environments, it had to be designed to the lowest common denominator: you cannot get lower than Unix.

The X Window System has been described in glowing terms as being *modular*, having a *layered architecture* and providing *hardware abstraction*. These are all considered generally desirable in operating system software. X is modular in the sense that the amorphous mass of X is not integrated in any way with the operating system. X is layered in the sense that X provides a lump with the low level functions, protocol handling and the graphics device driver while the applications have to provide all the "upper level" functions. X does not provide any form of hardware abstraction (See Box 5).

The most bizarre feature of X was that, as X could not be implemented "properly" as a system component on the lowest common denominator operating system, it was implemented as a application program accessing the display hardware directly.

This anomaly has persisted right through to the present: "X is unlike any other subsystem of Linux in that the hardware-accelerated video drivers it uses are located within the X server, which is outside the kernel ... normally Linux drivers and vital subsystems such as keyboard, USB, filesystem, serial I/O, et cetera are all located inside the kernel"[8].

25 years later, there are signs that the X windows architectural legacy is finally being eliminated, even if its egocentric minimalist design philosophy design lives on.

The major lesson to learn from X is that, although it was built using a design philosophy that set out to minimise the functionality and implementation effort in order to make it compact, efficient and quick to implement, X turned out oversized, painfully slow and very long in gestation.

Box 5 - Hardware abstraction in X and Unix

Hardware abstraction is one of the conventional requirements of an operating system. Its ensures that applications programs do not themselves need to support all possible peripherals that can be used for a particular function. X, however, works only with a single type of hardware (a frame buffer type of display controller). The lack of hardware abstraction was always a very serious problem with X, in particular its failure to support printers. Various fixes were attempted (such as combining X with display postscript) but the trauma caused by the arrival of Microsoft Windows, which did have hardware abstraction allowing pages to be output to a printer using the same application code as used for writing to a window, provoked the creation of a parallel system "XPrint". This was released in X11R6.3 in December 1996, a decade after the first release of X11 and later removed from X.Org Server in 2008 because "X is not an API". More to the point, X had never had an applications program interface (API) for any operating system, because Unix did not support hardware abstraction in any meaningful way.

Just a minute, you might say, there is a lot of Unix system documentation that states clearly that Unix provides hardware abstraction. **Rubbish!** Unix started off life pretending that all peripherals were paper tape reader / punches and then changed this to all peripherals being files, which is even worse.

At the time X was being designed, communication with a Unix system was almost exclusively by "glass teletype" terminals: "during the late 1970s and early 1980s, there were dozens of manufacturers of terminals including DEC, Wyse, Televideo, Hewlett Packard, IBM, Lear-Siegler and Heath, many of which had incompatible command sequences" (Wikipedia).

In order to use these terminals for anything other than Teletype (scrolling text) emulation, each type of terminal had its own "protocol" (command sequences). So before, you start editing a file with emacs or vi, for example, you needed to type

`set term=vt100`	for a DEC VT100
`set term=wyse60`	for a Wyse 60

Quite a lot of emacs manuals say that these commands "tell Unix the type of terminal being used". **Rubbish!** Unix does not know about terminals. These commands set an "environment variable" to be accessed by an application, so that the *application itself* can generate the right protocols for the terminal. This requires every application to have its own protocol handling for every terminal *supported by the application*.

The whole purpose of hardware abstraction is to provide a well defined common interface and have the right drivers installed in the operating system so that applications only need to implement a common interface and do not have to have their own device handlers for each type of peripheral. This requires an operating systems interface (or API) that directly supports the whole range of functions that a particular class of peripherals can provide - totally contrary to the Unix minimalist approach.

8 http://home.comcast.net/~fbui/

1986 - The rise and fall of RISC

Computer hardware design had its own equivalent of the software minimalisation theories but rather more successful.

In 1986, when the first commercial RISC (Reduced Instruction Set Computer) processors were shipped, the concept was already fairly old. From the mid 1970s it had been observed that the compilers of the time were often unable to take advantage of features intended to facilitate machine coding and that complex addressing inherently takes many cycles.

The argument was that such functions would better be performed by sequences of simpler instructions that could be execute faster (in a single cycle). RISC became synonymous with the use of uniform, fixed length instructions with arithmetic only in registers and dedicated load-store instructions to access memory.

The first production processors based on RISC principles were the Sun Microsystems SPARC processor, based on the Berkeley RISC project, and the rather different ARM from Acorn. These were followed by the Intel i860/i960, the IBM/Motorola Power processors, and MIPS, DEC Alpha, etc. These RISC processors all had dedicated workstation architectures built around them but looking back from 2009, it is clear that the technology has not been successful at displacing complex instruction set computer architectures. Of the large number of RISC architectures produced, only the SPARC and ARM are still holding their own.

What was wrong with the concept and why were the SPARC and ARM particularly successful?

The problem with the concept was that it was a one-size-fits-all theory, so while there is a general explanation of its failure (Box 6), the two successes had radically different explanations (Box 7).

In the case of the SPARC, Sun Microsystem's long standing opposition to the "Axis of Wintel" as a marketing ploy has weakened of late and their commitment to the SPARC is looking less and less firm. The June 2009 Top500 list of supercomputers gives two Sun clusters in the top 10: both are x86 architecture. There is every sign that the SPARC is on its way out, leaving the ARM as the only survivor.

Box 6 - The RISC of failure

The argument at the time that compilers were unable to take advantage of the instruction sets of the mainstream processors is a polite way of saying "the C language and the standard C compiler were designed for PDP7 computers". The "back ends" of these compilers converted primitive PDP7 type instructions into the nearest equivalent groups of instructions on the target computers. They therefore generated excessive and often redundant code for CISC (Complex Instruction Set) computers.

This still tends to be true, but more recent C compilers for complex instruction set computers include "optimisation" to identify groups of instructions that can be reduced to fewer instructions. A recent C compiler for a CISC processor will normally generate fewer instructions than a compiler for a RISC processor, thus giving an advantage to CISC architectures in terms of code size and, therefore, instruction memory bandwidth required.

The argument that an instruction that includes complex addressing will take more than one cycle to execute is still valid but it is illogical as a justification.

On a RISC processor, it will still take more than one cycle, because it requires separate instructions for the addressing. Technology advances since the 1970s, however, tilts the balance firmly in favour of CISC as, with a pipelined architecture the address calculations can be performed by a small dedicated shift / multiply / add / fetch addressing unit at an earlier stage of the pipeline so that addressing cycles overlap instruction execution cycles.

The real reason for the failure of RISC, however, was already apparent by the time that the first RISC machines appeared. The QL processor was memory bandwidth limited, not instruction execution time limited. This was generally true of 1984 workstation technology. The potentially higher instruction execution speed of a RISC processor would give no advantage. The fixed width, rather wide instructions and the use of several RISC instructions to do the job of one CISC instruction increased the memory bandwidth required to execute most operations: a great disadvantage.

In the short term, the smaller core of a RISC processor made it possible to add instruction caches with burst accesses to the main memory to compensate for the increased bandwidth required. It was not long, however, before technology advances made instruction caches practical for CISC processors as well. Because the CISC core took more space, there was less space for the instruction cache, but, because CISC code was more compact, less cache space was needed.

As processor speed improvements consistently outstripped memory speed improvements over the next decades, any possible advantages of RISC architectures for general purpose workstations just became more and more hypothetical.

Box 7 - The RISC of success

The SPARC

The SPARC processor owes its existence and its survival to just one company: Sun Microsystems.

In the early days, SPARC processors really did yield performance benefits in the C / Unix environment where they were used. These processors could be regarded as C / Unix optimised in two ways.

The primitive SPARC instructions were well suited to code generation by early C compilers

The processor has a dedicated register stack to mitigate the inefficiency of the C function call conventions (the C function call conventions require that parameter values cannot be modified - this is efficient and simple where the stack is used for passing parameters, as required for the PDP7, but very inefficient where values are passed in registers in more modern processors as it requires the register values to be saved and restored).

The SPARC only has a small hardware window onto the register stack, everything above the window has to be pushed out to, and retrieved from memory. The larger the window, the more efficient the execution, but, unfortunately the more costly the task switching as the whole window has to be saved. This was not a problem for earlier Unix versions as Unix was a multi-user system with very expensive, very slow and infrequent task switching. This became a problem when native threads were introduced in an attempt to pass Unix off as a multitasking system.

The ARM

The ARM architecture seems likely to have a much better long term future. The ARM instruction set really only qualifies as RISC from the point of view of being designed for single cycle instruction execution. With features such as generalised conditional execution of each instruction, one ARM instruction can sometimes replace two CISC instructions. It would be more appropriate to consider it not as a Reduced but as a Regular Instruction Set Computer. The ARM architecture escaped the fate of real RISC architectures for two other reasons.

The first was the licensing strategy which propelled it into the domain of embedded and integrated processors. For embedded applications with much less RAM than a typical workstation, low power, fast, static RAM gives far higher bandwidth than power hungry, slower, dynamic RAM. When executing from on chip RAM and ROM the external memory bus bandwidth is not a performance limiting factor, making the bandwidth inefficiency RISC instruction set less important, so the compact, low power fast ARM core provides a very competitive solution, particularly in custom and semi custom chip solutions.

The second reason that the ARM architecture has come to dominate portable embedded and integrated systems is that the last pretence of RISC was abandoned with the introduction of the Thumb instruction set that packs the most used 32 bit instructions into 16 bits, thus reducing the bus bandwidth problems as well as bringing the code size closer into line with CISC processors.

Strategy

by Stephen Poole

It would be very incomplete not to include a game of strategy in this present series of classic computers games for the QL. Strategy games generally consist of a territory which has to be secured against an adversary, with both tactics and chance deciding the outcomes of the conflict.

So I decided to try to design such a game as simply as possible, using two players so as not to have to program the QL using Artificial Intelligence, which would greatly lengthen the code.

The territory is simply a grid, whose size is determined by the difficulty chosen. Each player (green or red), in turn chooses a position to try to gain (by hitting an across 'x' then down 'y' figure), and the QL then decides which player has the most force for that spot, and draws the winning player's letter colour and force figure on the grid. (To find out how to augment your force factor, study the listing!).

If you choose a location that has already been contested, you can still win it if you obtain a higher force than that which your opponent has already in place. If both adversaries get the same force, the QL beeps and the game moves on to the next player.

The force of each player for each try is shown at the bottom of the screen. Maximum force is equal to the difficulty level per grid. Once maximum force has been reached, a position is won irrevocably. When one player has won the biggest share of territory, the game is over!

As mentioned previously, there is a winning strategy, but I shall give you no clues!

I found it very difficult to obtain a wide range of random figures, so to improve the randomness I used Mark Knight's True_Randomise function.
I did intend to write a 'wargame' to make the games series complete, but for philosophical reasons decided against this. Strategy is as near to war as I shall be getting!
Happy TacTicing.

```
100 ::
110 REMark Strategy_bas by S.Poole,
    v10apr2008. for QL Today
120  REMark beta-test by Bruno Coativy.
130 :
140 REMark Initialise:
150  CLEAR: OPEN#1,con_16: n$='23456789'
160  WINDOW 512,256,0,0: PAPER 0: INK 7: CLS
170   PRINT'Difficulty? (2 to 9)':
     i$=INKEY$(#1,-1)
180   IF i$ INSTR n$: d=i$: CLS: ELSE CLS: GO
     TO 170
190  DIM t(2,d,d): CSIZE 2,1: tot1=0: tot2=0
200   xit=INT((d*d)/2)+1: n$='1'&n$: n$=n$(1
     TO d)
210   True_RANDOMISE DATE
220 :
230 REMark draw grid:
240  FOR x=1 TO d
250      FOR y=1 TO d: AT y,x: PRINT'_'
260  END FOR x
270  FOR x=1 TO d: AT 0,x: PRINT x;: END FOR
    x: PRINT;'x'
280  FOR y=1 TO d: AT y,0: PRINT y : END FOR
    y: PRINT 'y'
290 :
300 REPeat loop
310    FOR player=2,4
320        INK player
330        IF tot1+tot2=d*d: DONE 11,0,'
           Draw...',0
340        AT 10,0: CLS 3: PRINT'player'!
           player/2
350        AT 10,9: PRINT'x:';: i$=INKEY$
           (#1,-1)
360        IF i$ INSTR n$: x=i$: PRINT;x: ELSE
           GO TO 350
370        :
380        AT 10,13: PRINT'y:';: i$=INKEY$
           (#1,-1)
390        REMark Hit '0' for Oops...
400        IF i$='0': GO TO 340
410        REMark Hit 'Q' to Quit game...
420        IF i$ INSTR 'Qq': STOP
430        IF i$ INSTR n$: y=i$: PRINT;y: ELSE
           GO TO 380
440        i$=INKEY$(#1,50)
450        :
460        REMark Get Each player's previous
           force:
470        t1xy=t(1,x,y): t2xy=t(2,x,y)
480        REMark Don't play if position
           taken:
490        IF t1xy=d OR t2xy=d: GO TO 340
500        rd1=RND(t1xy TO d): rd2=RND(t2xy TO
           d)
510        AT 11,1: INK 2: PRINT rd1;: INK 4:
           PRINT !rd2
520        REMark If forces equal, get next
           player:
530        IF rd1=rd2: BEEP 12345,67: GO TO
           770
540        :
550        REMark Player 1 is strongest:
560        IF rd1>rd2 THEN
570           REMark If best score, note it:
580            IF rd1>t1xy: t(1,x,y)=rd1
590           REMark Is score better than the
              other player's?:
600            IF rd1>t2xy: AT y,x: INK 2:
               PRINT rd1
610           REMark Score is maximum, so cell
              is won:
620            IF rd1=d: tot1=tot1+1
630            IF tot1=xit: DONE 11,0,'Bravo',
               1
640            REMark Same player retries:
650            player=2: GO TO 320
660        END IF
670        :
680        REMark Player 2 is strongest:
690        IF rd2>rd1 THEN
700           IF rd2>t2xy: t(2,x,y)=rd2
710           IF rd2>t1xy: AT y,x: INK 4:
              PRINT rd2
720           IF rd2=d: tot2=tot2+1
730           IF tot2=xit: DONE 11,0,'Bravo',2
740           player=4: GO TO 320
750         END IF
760         IF tot1+tot2=d*d: DONE 11,0,'
            Draw...',0
770    END FOR player
780 END REPeat loop
790 :
800 DEFine PROCedure DONE(dn,ac,d$,pl)
810  AT dn,ac: CLS 3: INK 7: PRINT
     d$!'player'!pl!'!'
820  AT 0,0: CLS 3: PRINT'Another? (y/n)?'
830  i$=INKEY$(#1,-1): IF i$=='n': STOP: ELSE
     RUN
840 END DEFine
850 ::
860 ::
870 REMark True_RANDOMISE_bas
880 REMark by Mark Knight
890 REMark QLT v1,i6, mar-apr97, p.15.
900 REMark RANDOMISE is only 16-bit!
910 :
920 DEFine PROCedure True_RANDOMISE(_32_bits)
930  LOCal adr,pk: adr=ALCHP(4)
940  IF adr<0: RANDOMISE: RETurn
950  POKE_L adr,_32_bits
960  :
970  REMark Get 16 bits only:
980  pk=PEEK_W(adr+2)
990  RANDOMISE pk
1000  RECHP adr
1010 END DEFine
1020 ::
```

QUANTA

Independent
QL Users Group

World-wide Membership is by subscription only,
Offering the following benefits:
Bimonthly Magazine - up to 52 pages
Massive Software Library - All Free!
Free Helpline and Workshops
Regional Sub-Groups. One near you?
Advice on Software and Hardware problems
Subscription just £14 for Full Membership

PayPal (see QUANTA Web Site),
Cash, Cheques and Postal Orders Accepted

Now in our Twenty Seventh Year

Further details from the Membership Secretary

John Gilpin, 181, Urmston Lane,
Stretford, Manchester, M32 9EH (UK).
Tel. +44 (0) 161 865 2872
Email: membership@quanta.org.uk

Visit the QUANTA Web Site
http://www.quanta.org.uk

Next QUANTA Sponsored Event
Annual General Meeting 2010 and Workshop

Date: Sunday 18th April 2010
Workshop from 10.00 a.m to 4.30 p.m: (Doors open from 9.00 a.m)
Annual General Meeting at 3.00 p.m prompt.

Venue: BRC Community Centre
"FIRCONE", 1237 Stratford Road, Hall Green, Birmingham, B28 9AA

Full details from
Chairman@quanta.org.uk

Event Details

The QL Show Agenda

Come to Vienna!

International QL-meeting 2010 in Prottes (near Vienna). Thursday, 3rd (bank holiday) to Sunday, 6th of June 2010.

Gerhard Plavec, the organiser, has already placed a lot of useful information on his website http://kuel.org (German, French and English)

He plans to turn the Saturday into the main day, but if the majority of visitors prefers to come on Friday, it can easily be changed.

On his website, you'll find already loads of information about accomodation (including staying at Gerhard's place), even with a tent ... he can also provide electricity for visitors who come with their motor homes. In both cases, please contact Gerhard in advance.

Prottes can be reached directly by car and train, and, of course, with every travelling method via Vienna (airport or ship, e.g. from Bratislava).

There are several tourist sites nearby (not to mention Vienna itself). The railroad museum in Strasshof (very close) will be open all four days, and they will even get steam engines going on Sunday. If there is enough interest, Gerhard may ask if they would do it on the main day (see above, most likely Saturday or Friday).

Gerhard can be contacted via email: gerhard.plavec@gmx.at or phone +43 699 81856765

We are informing you early to ensure you can fit the meeting in with other holidays or journeys ... so let's turn this event into a big event, kind of re-union of QLers who have not met for a long time!

J-M-S will most likely take part in the event on Friday or Saturday, whatever will be the main day. Sunday seems to be the day of the visit of the Railroad museum, as can be read on Gerhard's website...
J-M-S will bring back-issues of QL Today, some CDs and other stuff. If you are looking for something particular and want it to be brought and reserved for you, please let me know in advance, preferably via email: SMSQ@J-M-S.com
Marcel Kilgus plans to visit the event on the main day. Tony Tebby will come as well.
We all look forward to see many of you there!

The Next Issue

We plan to have the next issue ready for you at the Vienna QL Meeting as announced above. That's why we have an early deadline - as the usual shipping date would be mid/end of June, not beginning. Please send material as soon as possible, so that we can start early on producing this issue and ensure that it will be ready as planned.